Identification and clustering of seasonality patterns for demand forecasting

Simo Salmirinne

Helsinki May 29, 2020

Master's thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Simo Salmirinne

Työn nimi — Arbetets titel — Title

Identification and clustering of seasonality patterns for demand forecasting

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	May 29, 2020	48 pages + 0 appendix pages

Tiivistelmä — Referat — Abstract

Time series are essential in various domains and applications. Especially in retail business forecasting demand is a crucial task in order to make the appropriate business decisions. In this thesis we focus on a problem that can be characterized as a sub-problem in the field of demand forecasting: we attempt to form clusters of products that reflect the products' annual seasonality patterns. We believe that these clusters would aid us in building more accurate forecast models.

The seasonality patterns are identified from weekly sales time series, which in many cases are very sparse and noisy. In order to successfully identify the seasonality patterns from all the other factors contributing in a product's sales, we build a pipeline to preprocess the data accordingly. This pipeline consist of first aggregating the sales of individual products over several stores to strengthen the sales signal, followed by solving a regularized weighted least squares objective to smooth the aggregates. Finally, the seasonality patterns are extracted using the STL decomposition procedure. These seasonality patterns are then used as input for the k-means algorithm and several hierarchical agglomerative clustering algorithms.

We evaluate the clusters using two distinct approaches. In the first approach we manually label a subset of the data. These labeled subsets are then compared against the clusters provided by the clustering algorithms. In the second approach we form a simple forecast model that fits the clusters' seasonality patterns back to the observed sales time series of individual products. In this approach we also build a secondary validation forecast model with the same objective, but instead of using the clusters provided by the algorithms, we use predetermined product categories as the clusters. These product categories should naturally provide a valid baseline for groups of products with similar seasonality as they reflect the structure of how similar products are organized within close proximity in physical stores.

Our results indicate that we were able to find clear seasonal structure in the clusters. Especially the k-means algorithm and hierarchical agglomerative clustering algorithms with complete linkage and Ward's method were able to form reasonable clusters, whereas hierarchical agglomerative clustering algorithm with single linkage was proven to be unsuitable given our data.

Avainsanat — Nyckelord — Keywords
Time series, Unsupervised machine learning, Clustering, Seasonality decomposition

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

We live in the era of big data where massive amounts of information is constantly collected in many domains. This information comes in many shapes and sizes - for example, videos can be considered as ordered sequences of 2D images, news feeds as collections of strings and sales of a product over a period of time can be expressed as time series. Of all the data types, this last-mentioned time series is the most essential in this thesis.

Several fields of science utilize time series in their research. Models to detect signs of severe heart failures from heart rate time series have been developed in medicine [1] and accurate long-term weather forecasts are crucial in businesses such as weather derivatives [2]. Stock market volatility changes have been attempted to explain via economic variables through time [3] and utilizing time series to forecast product sales has been of interest since at least 1960s [4]. These are just a few examples of applications that make use of time series.

Retailers collect and produce large quantities of data every day, which can be utilized to make the appropriate business decisions. For example, demand estimates assist merchants and retailers to have the optimal quantities of items in their inventory at any given time. In this thesis we focus on a specific sub-problem that stems from the larger problem of estimating demand in retail business. We research how to automatically create clusters of products that reflect the products' annual seasonality patterns. These seasonality patterns are identified from annually repeating characteristics in the sales data. For instance, some products, such as ice cream or sunscreen, are known to peak in sales volume during the summer time. On the other hand, sales peaks of some other products can be bound to certain events, such as Easter or Christmas holidays. Especially, the Easter holiday can be problematic in many circumstances as its occurrence varies from year to year.

There exists a few approaches to somewhat similar studies in the literature. Kumar et al. [5] proposed a novel hierarchical clustering procedure with a distance function that accounts for measurement errors in the data. However, in their research the seasonality patterns were initially granted and ground truth of the data labels was presumed. Recent development by Jha et al. [6] attempts to cluster groups of items based on similarity of possibly sparse sales profiles and semantic features. They proposed a clustering procedure based on a local search heuristic with Spearman's correlation as their choice of dissimilarity. This clustering procedure was found to be competitive with several well-known clustering algorithms.

In our approach we first build a pipeline to estimate the seasonality of products based on weekly sales data. We begin by aggregating sales of individual products over several stores to strengthen the sales signal of these products. Then we formulate a weighted least squares objective using pricing information of the sales data to smooth the aggregated sales time series in order to reduce effects of non-seasonal factors. Then we use the STL decomposition method [7] to estimate the seasonality. After we have estimated the products' seasonal patterns, we pass these patterns as

input for the k-means algorithm and several hierarchical agglomerative clustering algorithms.

In reality we often have strong beliefs, or even ground truth, of how certain items sell given the time of year just based on intuition from our own experiences in every day life. This prior knowledge is used for validation of the cluster results. For example, ice creams are not likely to end up in the same clusters as certain Christmas chocolates. For a second validation method that reflects the larger demand estimation problem, we formulate a simple forecast model that attempts to fit the cluster's estimated seasonality patterns into the observed sales time series of individual products. Results provided by this forecast model are then compared against other forecast models.

One limitation of our approach is that estimating the periodicity of a signal in general requires the signal to be of several periods long. Therefore we limit the scope of this thesis such that we omit the analysis of products that do not meet the threshold of two consecutive years of sales. For instance, in e-commerce it is common that many items have a short lifespan whereas in brick and mortar stores there is a larger overhead in modifying the assortment. Our research focuses entirely on items that are sold in brick and mortar stores.

The results indeed suggest that the clustering algorithms including the preprocessing pipeline are able to form clusters that show clear seasonal structure. This becomes especially apparent when we compare the seasonality of the clusters against the seasonality of predetermined product categories, where the product categories reflect how similar items are organized within close proximity in physical stores. The ideas and approaches presented in thesis need not to be retail-oriented but should generalize equally well into other domains.

Section 2 works as an introduction to the key data type in this thesis: time series and its properties. In Section 3, we review the exploratory task of clustering in the field of unsupervised machine learning. We consider the advantages and limitations of some well-known clustering methods. Section 4 combines Sections 2 and 3 to provide general theory for the clustering of time series. Section 5 presents a real world application of clustering sales time series based on their similarity in seasonal characteristics. The last two Sections 6 and 7 analyze and conclude the results from several standpoints, and finally provide viable baselines for future research.

2 Time series

Every day we observe and analyze massive amounts of data in practically every domain. One of these data types that also incorporates the time of the observation is known as *time series*. A time series is a sequence of data points, typically observed at equal intervals in time. In this section we review the essential characteristics of time series, several commonly applied noise reduction procedures, time series decomposition methods and models.

2.1 Background

We define time series as an ordered sequence of pairs $(y_1, t_1), (y_2, t_2), ..., (y_n, t_n)$, where y_t denotes the observation at time t. When we associate a single observation per time interval, such as the hourly temperature level, the data type is known as *univariate time series*. If we also augment the hourly temperature level observations with, for example, the speed and direction of the wind, the data type is known as *multivariate time series*. Time series can either be discrete or continuous: observations can be taken at specific points in time or continuously through time. This thesis considers time series as univariate time series observed at intervals of equal length, unless specified otherwise.

2.2 Characteristics of time series

Time series often reveal meaningful characteristics in data dependent of time. For example, if we consider the observations of hourly temperature levels, we would most likely see a strong correlation between the observations that are a full day, or 24 hours, apart from one another. Expanding these observations over the last few decades, we would likely see similar correlation between the times of year, and probably even an increment in the long-term temperature levels due to global warming.

Figure 1 shows a plot of quarterly beer production in Australia between years 1955 and 2010. This data is part of the fpp2 package in the CRAN distribution[1]. We see a clear long-term increase in the production starting from 1960 going up to 1975, followed by a rather constant phase lasting until 1990, and finally a slight linear decrease up to 2010. This long-term change in the mean is typically called the *trend component* or *trend*. The jagged pattern that repeats itself at fixed intervals, or annually in this case, is called the *seasonality component* or *seasonality*. Intuitively, the annually repeating peaks in production during the third quarter can be explained by an increase in demand in the summer season in the southern hemisphere.

[1] https://cran.r-project.org/web/packages/fpp2/index.html

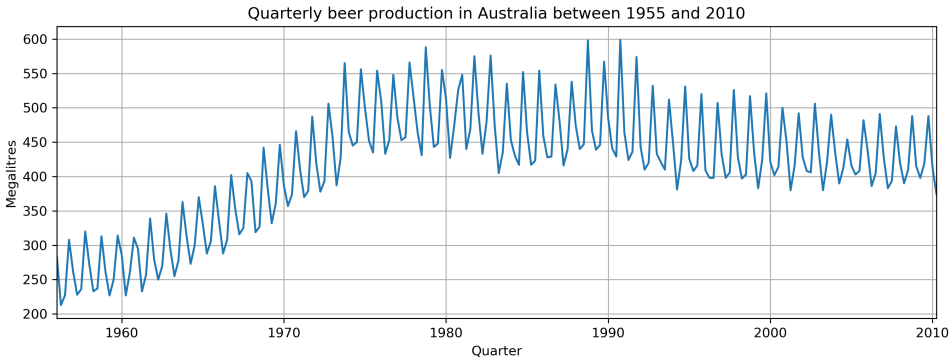

Figure 1: Example time series of the quarterly beer production in Australia. A clear increase in trend can be seen starting from the 60's going up to the mid 70's. There are also strong annual peaks in beer production during the third quarters to accommodate for the demand in the summer season in the southern hemisphere.

The components of time series can be summarized as follows:

- Trend: a long-term increase or decrease in the data. The trend can either be linear or non-linear and can change in time.
- Seasonality: predictable characteristics occurring in the data between intervals of fixed length in time.
- Remainder: what is left in the data after removals of the trend and seasonality components. In the literature the remainder is sometimes called the irregular or residual component.

Once we assume a meaningful decomposition of time series into separable components exists, we can express the time series as a summation or multiplication of these components. Let y_t be an observation of a time series at time t. We denote the *additive model* as a time series of the form

$$y_t = T_t + S_t + R_t, \tag{1}$$

where the T_t is the trend component, S_t is the seasonality component and R_t is the remainder component, all at time t. Respectively, we denote to the *multiplicative model* as a time series of the form

$$y_t = T_t \times S_t \times R_t. \tag{2}$$

In practice, we can convert a multiplicative model into an additive model simply as a log transformation of the multiplicative model, if necessary. In this thesis, our focus is on the additive models.

2.3 Time series smoothing

In many applications, the signal or the data is affected by various sources of interference and noise. Essentially we would like to to capture the relevant patterns and give little weight to the abnormalities in the data — for what is relevant, is often application specific. Here we present some commonly used concepts for time series smoothing from the areas of linear filtering, regression and curve fitting. These methods form the basis for many time series decomposition methods including the procedures used in our application in Section 5, which involves estimating seasonality patterns of typically noisy sales time series.

2.3.1 Background

Many of the smoothing procedures differ in several characteristics. These characteristics include simplicity and interpretability, computational performance, capability of estimating missing values, influence of individual points and bias near the endpoints. One simple family of smoothing procedures are *linear smoothers*. Let $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ be a sequence of real-valued observations at points $x_1, x_2, ..., x_n$. A linear smoother is an approximation $\hat{\boldsymbol{y}}$ that is a linear transformation of the observed values $\boldsymbol{y}$. More formally, we can express this linear transformation as $\hat{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y}$, where $\boldsymbol{A}$ is known as the *projection* or *smoother matrix*. A computationally tractable property of linear smoothers is that $\boldsymbol{A}$ does not depend on $\boldsymbol{y}$. For a review of linear smoothers, we refer to [8].

2.3.2 Moving averages

One widely used smoothing procedure is the *moving average*. Moving average, also known as the *running* or *rolling mean*, is a method that iteratively slides through the observations $y_1, y_2, ..., y_n$ giving each observation an estimate $\hat{y}_i$ as the mean of observations within the *neighborhood* of y_i. This neighborhood is known as the *span* (also called *window size* or *bandwidth*), and we denote the length of the span by m. The moving average of span m, or m-MA, is then denoted by

$$\hat{y}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}, \tag{3}$$

where $k = \lceil \frac{m-1}{2} \rceil$.

The moving average gives sufficient estimates of the long-term variations in the data. This property makes it one of the basic building blocks for many time series decomposition procedures. On the other hand, the moving average is effectively affected by irregularities in the data such that the neighborhood of an outlier is pulled towards the outlier. Depending on the objective, similar non-linear smoothing procedures, such as the moving median, may be preferred.

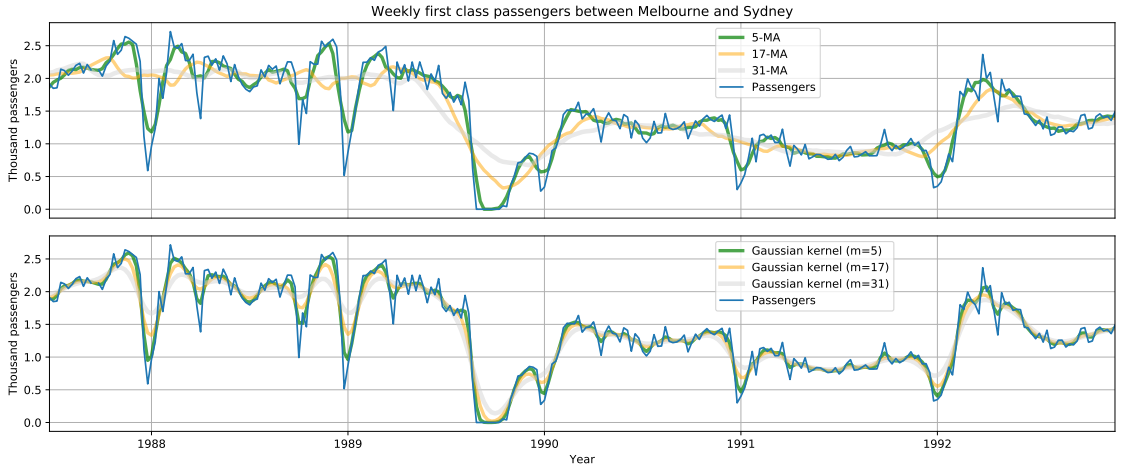

Figure 2: Two closely related transforms to the weekly first class passenger counts between Melbourne and Sydney in the early 90s. The sharp blue line denotes the passenger counts in thousands. Top: three parameterizations of the moving average transform ($m = 5$, $m = 17$ and $m = 31$). Bottom: three parameterizations of the Gaussian kernel ($m = 5$, $m = 17$ and $m = 31$). This data is part of the melsyd data set in the fpp2 package.

Figure 2 shows three moving average ($m = 5$, $m = 17$ and $m = 31$) transforms to the weekly first class passengers between Melbourne and Sydney in the early 90s (top). We see how the neighborhood of the moving averages is affected by rapid changes in the data, especially near the strong declines at the turns of the year. From the long-term estimation perspective, the 17-MA seems to reflect the long-term fluctuations the best, as the 5-MA is heavily affected by local changes and the 31-MA reacts to the long-term variations too slowly.

The moving average can also be expressed as a linear transformation. For example, a 3-MA has the following form

$$\hat{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \dots & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} \tag{4}$$

Note, that the top and bottom rows of $\boldsymbol{A}$ have been rescaled to make the row weights sum to unity. This rescaling ensures that the mean of the series remains unchanged.

2.3.3 Kernel smoothers

Another common family of linear smoothers are the *kernel smoothers*. Given a set of data points $(x_1, y_1), ..., (x_n, y_n)$, kernel smoothers are inherently symmetrically weighted moving averages, where the weight of individual values y_i decreases as the distance $|x - x_i|$ increases. The weights are assigned by a kernel function $K(x)$. Here a kernel function is a non-negative symmetric function that integrates to one over its domain. More formally, K satisfies the following conditions:

$$K(x) \geq 0, \tag{5}$$
$$K(x) = K(-x) \tag{6}$$

and

$$\int_{-\infty}^{+\infty} K(x)dx = 1. \tag{7}$$

In general, a kernel smoother has the following form

$$\hat{f}(x) = \sum_{i=1}^{n} w(x, x_i) \cdot y_i, \tag{8}$$

where $w(x, x_i)$ defines the sequence of weights for each data point. The weight function w is defined as

$$w(x, x_i) = \frac{K\left(\frac{|x-x_i|}{m}\right)}{\sum_{j=1}^{n} K\left(\frac{|x-x_j|}{m}\right)}, \tag{9}$$

where m is the bandwidth.

Perhaps the best known kernel function is the Gaussian kernel

$$K(x) = \frac{1}{\sqrt{2\pi}} \, \mathrm{e}^{\frac{-x^2}{2}}. \tag{10}$$

Figure 2 shows three different transforms to the passenger count data using a Gaussian kernel with bandwidths $m = 5$, $m = 17$ and $m = 31$ (bottom). We see that the transforms follow the original data quite closely, which is controlled by the bandwidth parameter. A Gaussian kernel effectively turns into a moving average smoother as $m \to \infty$.

2.3.4 Least squares smoothing

Given a sequence of observations $\boldsymbol{y}$, we can formulate a smoothing procedure as a least squares problem. Recall that in the general setting, a set of linear equations

$\boldsymbol{Ax} = \boldsymbol{b}$ can be approximated as a least squares problem by minimizing the sum of squared residuals objective

$$\|\boldsymbol{b} - \boldsymbol{Ax}\|_2^2, \tag{11}$$

where $\boldsymbol{A}$ is a matrix of the coefficients, $\boldsymbol{x}$ is a column vector of the unknowns and $\boldsymbol{b}$ is a column vector of the constants. This objective can be solved analytically (given that the columns of $\boldsymbol{A}$ are linearly independent) by

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}. \tag{12}$$

Furthermore, let $\boldsymbol{y}$ be the set of observations and $\boldsymbol{x}$ the set of smoothed observations that we wish to approximate. If we set $\boldsymbol{A}$ to be the identity matrix $\boldsymbol{I}$ and $\boldsymbol{b}$ the set of observations $\boldsymbol{y}$, the objective to be minimized becomes

$$\|\boldsymbol{y} - \boldsymbol{x}\|_2^2, \tag{13}$$

which has the trivial solution of setting $\boldsymbol{x} = \boldsymbol{y}$. If we would like $\boldsymbol{x}$ also to be smooth, or equally $x_i \approx x_{i+1}$, we can append the objective (13) with a secondary objective of *first order differences* $\boldsymbol{D}_1$, which should also be small.

Now the objective to be minimized becomes

$$\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{D}_1\boldsymbol{x}\|_2^2, \tag{14}$$

where λ controls the weight of adjacent values in $\boldsymbol{x}$ being as close as possible.

The first order difference matrix $\boldsymbol{D_1} \in \mathbb{R}^{(n-1)\times n}$ is denoted by

$$\boldsymbol{D}_1 = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix}. \tag{15}$$

The extended objective (14) has the closed-form solution

$$\hat{\boldsymbol{x}} = (\boldsymbol{I} + \lambda \boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{y}. \tag{16}$$

Respectively, another commonly used secondary objective in practice can be formed using the *second order difference* matrix $\boldsymbol{D_2} \in \mathbb{R}^{(n-2)\times n}$, which is denoted by

$$\boldsymbol{D}_2 = \begin{bmatrix} 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -2 & 1 \end{bmatrix}. \tag{17}$$

The second order difference matrix reflects the finite second order derivative as the difference of differences $(x_{i+1} - x_i) - (x_i - x_{i-1})$. The intuition is to enforce similarity in slope between adjacent values.

Figure 3 shows different approximations $\boldsymbol{x}$ as a minimized least squares approximation of $\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{D}_1 \boldsymbol{x}\|_2^2$, where the passenger count data is denoted by $\boldsymbol{y}$ (bottom).

2.3.5 Local regression

Local regression approximates $y_i \in \boldsymbol{y}$ iteratively by fitting a low-degree, typically linear or quadratic, polynomial given the m nearest neighbors of y_i. A common local regression method is LOESS (Locally estimated scatterplot smoothing) [9], where each point in the neighborhood of y_i is given a weight based on the distance to the neighbor $|x - x_i|$. Similar to kernel regression, the weights decrease as the distance to the neighbor increases.

In our smoothing scheme, the weighted least squares objective extends the least squares objective (13) by giving each residual a positive weight w_i. The weights are usually chosen to influence the approximated solution in a desirable manner or to assist the approximation using prior knowledge of the data. The weighted least squares objective is

$$\left\| \boldsymbol{W}^{1/2} (\boldsymbol{x} - \boldsymbol{y}) \right\|_2^2, \tag{18}$$

where $\boldsymbol{W}$ is a diagonal matrix with the weights w_i for each residual on the diagonals. Note that giving each residual a weight of one renders the weight matrix $\boldsymbol{W}$ to be the identity matrix and the objective becomes the same as the least squares objective (13).

A disadvantage is that the model assumes that the weights are known prior to fitting to the data. In the later chapters we attempt to iron out aberrant peaks in sales data as a weighted least squares objective using knowledge of rapid declines in the corresponding sales prices.

If we assume a linear model of the form

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}, \tag{19}$$

where $\boldsymbol{X}$ is the design matrix and $\boldsymbol{\beta}$ are the coefficients,

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \tag{20}$$

we can estimate $\boldsymbol{y}$ as a weighted least squares problem given by

$$\left\| \boldsymbol{W}^{1/2} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \right\|_2^2. \tag{21}$$

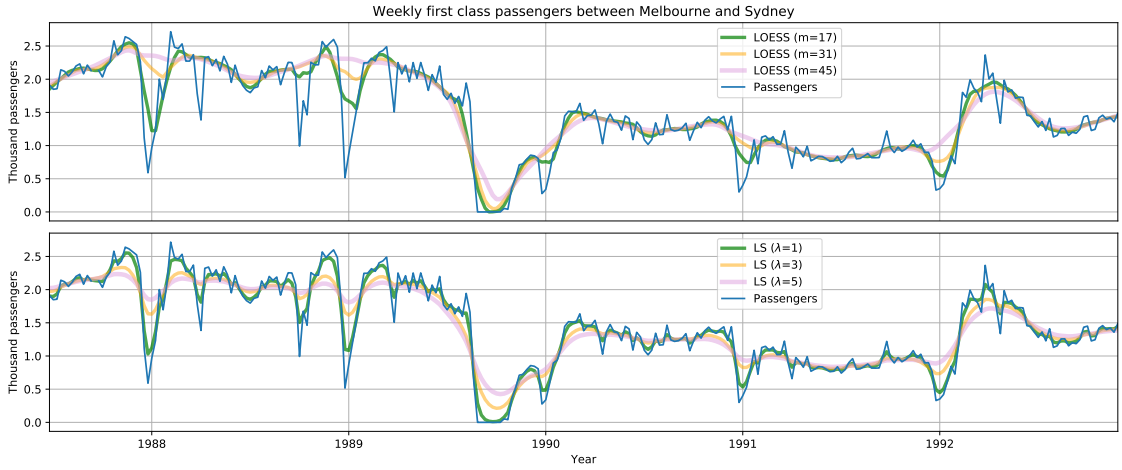

Figure 3: Similar to Figure 2, the passenger counts are denoted by the blue line. Top: Three different LOESS fits to the passenger count data with bandwidths $m = 17$, $m = 31$ and $m = 45$. Bottom: Three least squares approximations to the passenger counts, where λ controls the weight of adjacent values in the approximation being close to one another.

Analytic solution for the objective (21) is given by

$$\hat{\boldsymbol{\beta}} = \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{y}. \tag{22}$$

LOESS minimizes the weighted sum of squared residuals in (21) locally in the neighborhood of y_i. The weights are given by

$$w(x, x_i) = T\left(\frac{|x - x_i|}{m}\right), \tag{23}$$

where T is the tricubic function

$$T(u) = \begin{cases} (1 - |u|^3)^3, & \text{if } |u| < 1 \\ 0, & \text{if } |u| \geq 1. \end{cases} \tag{24}$$

Figure 3 illustrates the effect of the neighborhood bandwidth by fitting three locally linear LOESS curves ($m = 17$, $m = 31$ and $m = 45$) to the passenger data (top). We see that especially $m = 45$ gives a fairly robust estimation of the long-term variations without being affected by rapid changes in the data.

2.4 Time series models

Time series can be thought as a realization of a stochastic process. Statistical models that depict these stochastic processes are known as *time series models*. Several time

series models have been developed to model various processes and events, such as weather, stock or demand of consumer goods. In the following sections we present a brief introduction to some of the most popular time series models. For more thorough references, we refer to [10, 11, 12].

2.4.1 Background

Time series forecasting is arguably the greatest motivation for time series models. Say we are given a sales time series of the past three years of sales and we would like to forecast sales one year into the future. Here we could fit a time series model to the observed sales and try predict future sales based on that model. The usages of time series models are not limited to forecasting but are also applicable to tasks such as clustering. Model-based time series clustering approaches are discussed briefly in Section 4. In order to introduce the time series models, we first review some notation and definitions that incorporate temporal dependency.

A time series is called *stationary time series* if its values do not depend on the time of the observed value. In other words, a time series is stationary if the distribution of $(y_t, ..., y_{t+k})$ does not depend on t for all k. Some time series models assume the modeled time series to be stationary.

Differencing is a transformation of a time series given by computing the differences of consecutive observations. For example, first-order differencing of y is calculated as

$$y_t^{(1)} = y_t - y_{t-1}, \tag{25}$$

where we denoted the order of differencing as a superscript of y_t. Similarly, second-order differencing is defined as

$$\begin{aligned} y_t^{(2)} &= y_t^{(1)} - y_{t-1}^{(1)} && (26)\\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) && (27)\\ &= y_t - 2y_{t-1} - y_{t-2}. && (28) \end{aligned}$$

The concept of differencing relates to exactly the same idea of temporal proximity as with the first and second order difference matrices used to regularize the least squares objectives in Section 2.3.4.

One notational convenience to denote differencing is the *lag operator* or the *backshift operator*. It can be used to denote differencing without denoting differencing of the observation itself. For example, first-order differencing can be represented using the lag operator L as

$$\begin{aligned} y_t^{(1)} &= y_t - y_{t-1} && (29)\\ &= y_t - Ly_t && (30)\\ &= (1 - L)y_t. && (31) \end{aligned}$$

Respectively, the general case of dth-order differencing can denoted by

$$y_t^{(d)} = (1 - L)^d y_t. \tag{32}$$

2.4.2 ARMA models

Autoregressive moving average (ARMA) model is a combination of two models that rely heavily on the ideas of linear regression and differencing. Next we define these two models in order to formulate a complete ARMA model.

An *autoregressive model* AR(p) refers to a model that predicts values of an output variable y_t given p past values of the output variable itself. Moreover, the model assumes that the predicted value depends linearly on the past values and on some imperfectly predictable, *stochastic term*. The autoregressive model of order p is defined as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{33}$$

$$= c + \sum_{i=1}^{p} \phi_i y_{t-1} + \epsilon_t, \tag{34}$$

where c is a constant, $\phi_1, ..., \phi_p$ are the parameters of the model and ϵ_t is the stochastic term, or *white noise*. Moreover, ϵ_t is assumed to be sampled from the normal distribution with zero mean and σ^2 variance.

A *moving average* MA(q) model assumes that the output variable depends linearly on the residuals of the current and past values. The moving average model of order q is defined as

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}, \tag{35}$$

where μ is the mean of the time series, ϵ_t is white noise and $\theta_1, ..., \theta_p$ are the parameters of the model.

Note, that the n-MA smoothing method and the MA(q) model are only analogous in their names and the two should not be confused with one another.

Finally, the autoregressive moving average model combines the q autoregressive terms (34) and p moving average terms (35) into a single ARMA(p, q) model, which is defined as

$$y_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i y_{t-1} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}. \tag{36}$$

2.4.3 ARIMA models

Non-seasonal ARIMA (Autoregressive moving integrated average) models extend the ARMA models with a notion of *integration*. In this context integration means is the opposite operation of differencing. An ARIMA(p, d, q) model is defined as

$$y_t = c + \phi_t y_{t-1}^{(d)} + ... + \phi_t y_{t-p}^{(d)} + \theta_i \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t, \tag{37}$$

where d denotes the number of first-differences involved.

ARIMA models can also be extended to model seasonal time series with additional parameterizations P, D and Q, where these parameters are analogous to p, d and q with the exception that they model seasonally lagged values of the time series. In the literature these models are called seasonal ARIMAs or SARIMAs.

2.5 Decomposition methods

We are often interested in the individual components contributing to the observed time series. Various decomposition methods exist for time series. These methods can differ, for example, in their ability to handle outliers and missing values. Most methods allow arbitrary seasonal frequencies, but some methods, such as X-11 [13] and SEATS (Signal Extraction in ARIMA Time Series) [14], require the frequency to be defined from a fixed set of values. In this section we review a classical decomposition method based on moving averages, followed by a more advanced decomposition method that relies on local regression methods. Both of these methods allow seasonal frequencies of arbitrary length.

2.5.1 Classical decomposition

The classical, or naive, decomposition method dates back to the early 20th century and it has worked as a starting point for more advanced methods over the years [12]. This decomposition method estimates the trend component with moving averages. For demonstration, let y be a daily time series with 20 full weeks of data ($n = 140$) and weekly periodicity ($n_{(p)} = 7$). The decomposition of y into trend, seasonal and remainder components works as follows:

1. *Detrending.* The trend component $\hat{t}$ is calculated by $n_{(p)}$-MA and the *detrended* series y' is simply given by $y' = y - \hat{t}$. For the daily data with weekly periodicity, the moving average has a bandwidth of one week.

2. *Cycle-subseries averaging.* The *cycle-subseries* is a sequence of values constructed by iterating over the detrended series period by period, and taking the value from each period at the same position within the period. For the daily data, this corresponds to constructing one cycle-subseries by taking the value of each Monday, another cycle-subseries by taking the value of each Tuesday, etc. The cycle-subseries are denoted by c_i for $i \in \{1, ..., n_{(p)}\}$. Once the cycle-subseries have been constructed, the mean is taken from each cycle-subseries to obtain $\bar{c}_i$ for $i \in \{1, ..., n_{(p)}\}$.

3. *Normalization of the cycle-subseries means.* Next the cycle-subseries means are normalized such that the sum of the normalized cycle-subseries equals to zero. In regard to the daily data, this equals to the means of the days of the week summing to zero. The calculation is simply done by subtracting the

mean of the cycle-subseries means, $1/n_{(p)} \sum_{i=1}^{n_{(p)}} \bar{c}_i$, from each cycle-subseries mean.

4. *Period concatenation.* A single period $\hat{C}$ of length $n_{(p)}$ is obtained by concatenating the normalized means, $\hat{C} = (\bar{c}_1, ..., \bar{c}_{n_{(p)}})$. For the daily data, this corresponds to concatenating the normalized means of all Mondays, all Tuesdays, and so forth, to obtain a full week of data. The seasonal component $\hat{s}$ is then estimated by simply concatenating $\hat{C}$ with itself to make it of length n.

The remainder component $\hat{r}$ is calculated by detrending and deseasonalizing the original series, $\hat{r} = y - \hat{t} - \hat{s}$.

The advantages of the classical decomposition method include its simplicity, efficiency and interpretability. The downsides consist largely of the method being fragile to outliers in the data. Sometimes the seasonality itself can change in time, which the classical decomposition is unable to capture either.

Figure 4 shows the extracted components of the weekly first class passengers data set using the classical decomposition method with $n_{(p)} = 52$. In this decomposition, the trend component is clearly affected by the zero passenger counts in late 1989. Especially the remainder component seems to contain some structure (e.g. autocorrelation), which generally means that a more suitable decomposition should exist.

2.5.2 STL decomposition

One of the more robust decomposition methods is the Seasonal and Trend decomposition using LOESS (STL) [7]. STL is a computationally efficient and robust method for decomposing time series. The advantages of STL include computational efficiency, ability to decompose time series with missing values, robustness to outliers and flexibility in specifying the amounts of variation in the extracted components. For brevity, we attempt to outline the components of the STL procedure at a minimal level. More detailed references include [7] and [15].

There are two key components in the STL procedure: the inner and outer loops. The inner loop is the workhorse of the procedure. It iteratively recomputes estimates for the trend and seasonal components. The outer loop is an optional step to downweight aberrant points in the data. This downweighting is determined by certain statistical properties that are introduced in more detail later. STL uses both moving averages and LOESS for its smoothing operations.

We demonstrate the STL algorithm with the same fictional data that we used in the classical decomposition demonstration in Section 2.5.1: a daily time series y with 20 full weeks of data ($n = 140$) and weekly periodicity ($n_{(p)} = 7$). STL contains several parameters. These parameters are the seasonal periodicity $n_{(p)}$, number of iterations of the inner and outer loops, $n_{(i)}$ and $n_{(o)}$, and smoothing parameters $n_{(s)}$, $n_{(l)}$, $n_{(t)}$, $\lambda_{(s)}$, $\lambda_{(l)}$ and $\lambda_{(t)}$, which become apparent subsequently. Parameters of the

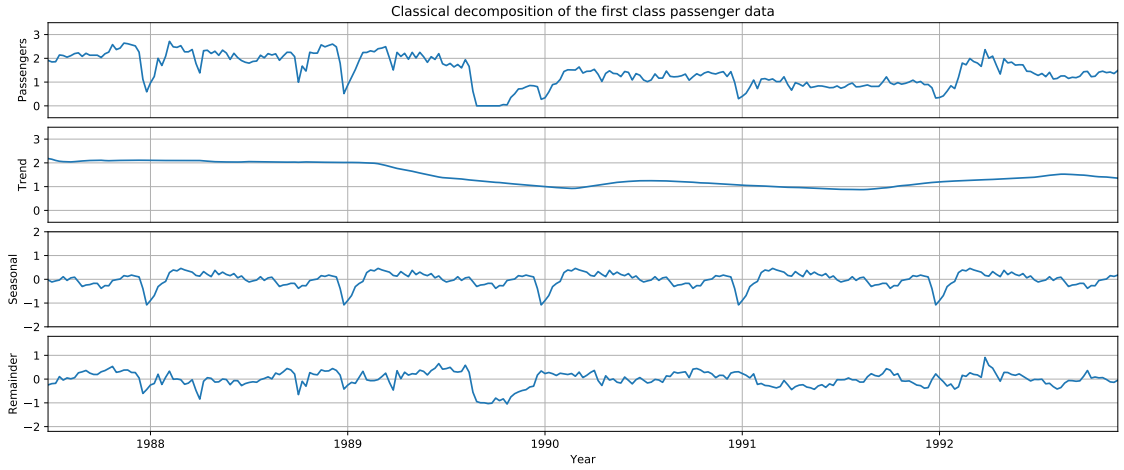

Figure 4: A decomposition of the weekly first class passenger data using the classical decomposition method with $n_{(p)} = 52$. Top panel: first class passengers in thousands. Second panel: estimate of the trend component based a on moving average. We see that the estimate is effectively dragged down near the zero passenger counts in late 1989. Third panel: the seasonal component that is repeated annually. Bottom panel: the remainder component that is obtained after removing the seasonal component from the detrended time series. All figures are on the same scale.

LOESS smoothers, bandwidth and degree of the local polynomial fit, are denoted by q and λ, respectively. Next we introduce the inner loop, which has the following steps:

1. *Detrending.* Iteration k begins by detrending the series given the trend component $\hat{t}^{(k-1)}$ computed in the previous iteration $k-1$ (unless this is the first iteration, for which this step is skipped). The detrended series is calculated as $y' = y - \hat{t}^{(k-1)}$. Values that are missing in y are also missing in y'.

2. *Cycle-subseries smoothing.* Next a total of $n_{(p)}$ cycle-subseries are constructed similar to how they were constructed in the classical decomposition procedure. Each cycle-subseries is then smoothed using LOESS ($q = n_{(s)}, \lambda = \lambda_{(s)}$). Smoothed values are computed at all positions, including positions of missing values, and also one position prior to the first value in the cycle-subseries and one position after the last value in the cycle-subseries. Lastly, the smoothed cycle-subseries are recombined chronologically to form a series $c^{(k+1)}$. The combined series has a length of $n + 2n_{(s)}$.

   For the 20 weeks of daily data, consider the cycle-subseries of all Mondays, denoted by c_1. After the LOESS smoothing, one extra value is added before the value of the first Monday and one extra value is added after the value of the last Monday. Therefore c_1 would have a length of 22. Similarly $c^{(k+1)}$ would have a length of 154.

3. *Low-pass filtering smoothed cycle-subseries.* This step estimates low-frequency

variation in the smoothed cycle-subseries, which is done by passing $c^{(k+1)}$ through a low-pass filter. This low-pass filter consist of applying the following smoothers sequentially: $n_{(p)}$-MA, $n_{(p)}$-MA, 3-MA and LOESS ($q = n_{(l)}, \lambda = \lambda_{(l)}$). Here the moving averages are considered to apply equally weighted moving windows in the sense that given an odd bandwidth integer m, the output of the smoother is missing $\lfloor m/2 \rfloor$ values at both ends. The output of the low-pass filter is denoted by $d^{(k+1)}$.

Regarding the daily data and the behavior of the moving averages, we see that exactly $n_{(p)} = 7$ values are dropped from both ends of the series $c^{(k+1)}$ by the three moving average smoothers. For example, given the first period in $c^{(k+1)}$, the first $n_{(p)}$-MA drops the values from Monday to Wednesday, followed by another $n_{(p)}$-MA dropping the values from Thursday to Saturday. Finally the 3-MA drops the value of Sunday. Thus the extra periods that occurred in $c^{(k+1)}$ are discarded in $d^{(k+1)}$.

4. *Detrending of smoothed cycle-subseries.* This step prevents the low-frequency variation from entering the seasonal component by subtracting the output of the low-pass filter $d^{(k+1)}$ from the smoothed cycle-subseries $c^{(k+1)}$. The seasonal component is then given by $\hat{s} = c^{(k+1)} - d^{(k+1)}$, where the first and last periods in $c^{(k+1)}$ are discarded.

5. *Deseasonalizing.* Deseasonalized series y^* is obtained by subtracting the seasonal component from the original series, $y^* = y - \hat{s}$. Values that are missing in y are also missing in y^*.

6. *Trend smoothing.* The deseasonalized series y^* is smoothed using LOESS ($q = n_{(t)}, \lambda = \lambda_{(t)}$) to obtain the trend component $\hat{t}$. Smoothed values are computed at all positions, including positions that contain missing values.

After $n_{(i)}$ iterations of the inner loop, the remainder component $\hat{r}$ is calculated by detrending and deseasonalizing the original series, $\hat{r} = y - \hat{t} - \hat{s}$.

Next we introduce the outer loop. If the series is presumed or detected to contain outliers, we may want to downweight the impact of these observations. This downweighting is done by calculating *robustness weights*, which are passed to the subsequent iteration of the inner loop to be used as multipliers for the LOESS smoother weights. The robustness weights p_i are calculated as

$$p_i = B\left(\frac{|\hat{r}_i|}{6 \cdot \text{median}(|\hat{r}|)}\right), \tag{38}$$

where B is the bisquare function

$$B(u) = \begin{cases} (1 - |u|^2)^2, & \text{if } |u| < 1 \\ 0, & \text{if } |u| \geq 1. \end{cases} \tag{39}$$

We see that elements of the remainder component, whose absolute value significantly differs from the median of absolute values in $\hat{r}$, are assigned weights close to zero.

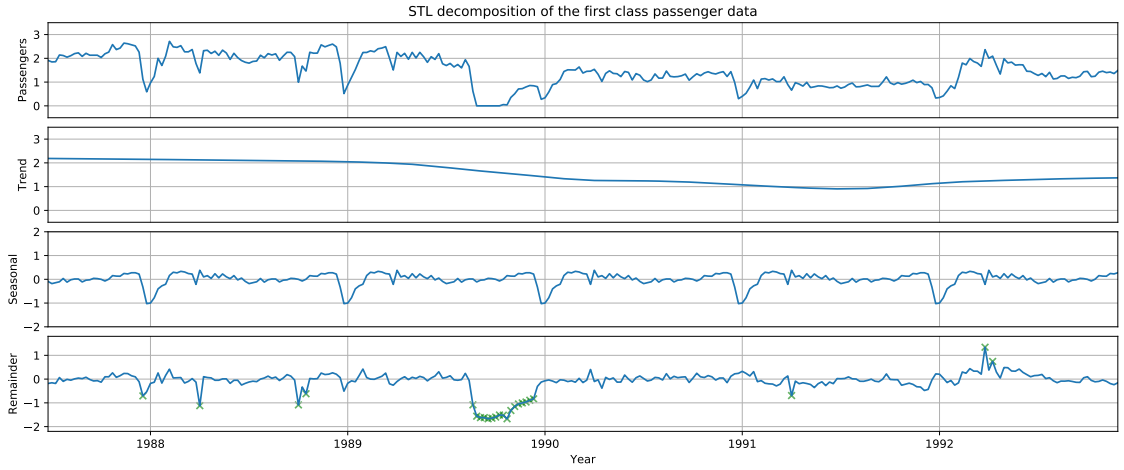

Figure 5: A decomposition of the first class passenger data using the STL decomposition method ($n_{(p)} = 52, n_{(s)} = 2831, n_{(t)} = 79, n_{(l)} = 53, n_{(i)} = 1, n_{(o)} = 15, \lambda_{(s)} = 0, \lambda_{(t)} = 1, \lambda_{(l)} = 1$). Top panel: weekly first class passengers in thousands. Second and third panel: iteratively computed estimates of the trend and seasonal components based on local regression. Bottom panel: the remainder component. Points that were given zero weight by the last outer loop are shown as green crosses. Here these points are effectively outliers.

Whereas values close to the median are assigned values close to one, thus having relatively small effect when multiplied with the LOESS smoother weights. After each iteration of the outer loop, the inner loop is run for $n_{(i)}$ iterations. The outer loop is run for a total of $n_{(o)}$ iterations.

Figure 5 shows the extracted components of the weekly first class passengers data set using the STL decomposition. We see that the trend extraction looks fairly sufficient, even though there is a slight bump in the passenger counts in the early 1992, which could be interpreted as a rapid increase in the trend. A rule of thumb would be to decrease the parameter $n_{(t)}$ to allow more variation in the trend component.

While the seasonal component in Figure 5 seems static, a close look at the periodic behaviour in the data reveals that the troughs in passenger counts near the turns of the year actually decrease in time. This can be explained by the parameter $n_{(s)}$ being much larger than n, which practically renders the cycle-subseries LOESS smoother in step 2 into a moving average. To account for dynamic seasonality, we could set a much lower $n_{(s)}$, say $n_{(s)} \approx n$.

2.6 Measuring the strength of the components

In general, we can divide the task of measuring the strength of the time series components into two categories. The first category consists of methods for detecting unknown periodicity. If we are given some temporal data, the task would be to explore if the data follows, for example, daily, weekly, or annual periodicity, or

whether there is any periodicity in the data at all. This problem has well-studied approaches such as [16, 17]. The second category consists of methods for measuring components that are already known.

Given a time series decomposition into trend $\hat{t}$, seasonality $\hat{s}$ and remainder $\hat{r}$, Wang et al. propose two intuitive formulas for measuring the strength of the trend and seasonality components [18]. The strength of the trend component is computed as

$$f_t = \max\left(0, 1 - \frac{\text{var}(\hat{r})}{\text{var}(\hat{t} + \hat{r})}\right), \tag{40}$$

which assigns f_t a value between 0 and 1. If the series has a strong trend, the variation of the deseasonalized data $\hat{t} + \hat{r}$ should be much larger than the variation of the remainder component $\hat{r}$, thus $\text{var}(\hat{r})/\text{var}(\hat{t}+\hat{r})$ should be relatively small, and f_t should be assigned a value close to 1. Respectively, the strength of the seasonal component is computed as

$$f_s = \max\left(0, 1 - \frac{\text{var}(\hat{r})}{\text{var}(\hat{s} + \hat{r})}\right). \tag{41}$$

3 Clustering

Often when we observe new data, we have very little prior knowledge of the process generating the data points. Clustering is an unsupervised machine learning task for assigning unlabeled data points into homogeneous groups, called clusters. Homogeneity of a cluster is typically expressed by minimizing some measure of dissimilarity between the members of the cluster. The uses of clustering are not limited to data exploration tasks but are also closely related to tasks such as lossy compression in the form of vector quantization [19]. In this section we review general clustering theory, measures of similarity, various clustering methods and cluster evaluation techniques.

3.1 Background

Based on the strategy of assigning the data points into clusters, clustering algorithms can be split into *hard* and *soft* (sometimes called *fuzzy*) clustering algorithms. Hard clustering algorithms assign each data point into exactly one cluster, whereas soft clustering algorithms define a distribution of assignments for each data point into each cluster. Han et al. [20] make a distinction of high-level clustering strategies into the following categories: partitioning methods, hierarchical methods, density-based methods, grid-based methods and model-based methods. In this thesis we focus primarily on the partitioning and hierarchical clustering methods.

3.2 Similarity measures

The choice of similarity is a crucial decision in any classification or clustering task. Depending on the objective, we may wish to denote similarity, for example, as similarity between distinct attributes or correlation between several attributes. Similarity between data points typically expressed as *data matrices* or *dissimilarity matrices*. A data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, where n is the number of the data points and p is the number of the attributes or features, is a matrix that has the data points on its rows and the attributes on its columns. For example, $\boldsymbol{X}_i$ represents one data point in the data matrix. On the other hand, a dissimilarity matrix $\boldsymbol{D}^{n \times n}$ represents pair-wise dissimilarity between two data points on the cells, such that $\boldsymbol{D}_{i,j}$ represents the dissimilarity between data points i and j. A dissimilarity matrix can always be computed from a data matrix, but not the other way around.

Dissimilarity between data points is often measured by *metrics*. Formally a metric d given set X is defined as a function $d : X \times X \to [0, \infty)$ given that the following

conditions are satisfied:

$$d(x, y) \geq 0 \tag{42}$$
$$d(x, y) = 0 \Leftrightarrow x = y \tag{43}$$
$$d(x, y) = d(y, x) \tag{44}$$
$$d(x, z) \leq d(x, y) + d(y, z). \tag{45}$$

The most used distance metrics used to measure similarity of data points are probably the variations of the Minkowski distance. We denote the Minkowski distance between points $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ as a metric

$$d(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{p} |a_i - b_i|^q \right)^{1/q}. \tag{46}$$

Particularly, the case of $q = 2$ is called the Euclidean distance. Another common variation of the Minkowski distance is the Manhattan distance ($q = 1$), which is also known as the city block distance. Note that Minkowski distances with $q < 1$ are not formally metrics as they violate the triangle inequality (45).

Sometimes the dissimilarity is calculated by distance measures that are not formally metrics, i.e. they violate the symmetry property (44) $\boldsymbol{D}$. Non-symmetric dissimilarity matrices can be converted symmetric by $(\boldsymbol{D} + \boldsymbol{D}^T)/2$.

There lies one possible issue with the variations of the Minkowski distances that should be carefully considered. In high dimensional spaces, we are often faced with a phenomena known as the *curse of dimensionality*. This is a result of these measures becoming less meaningful in higher dimensions as when the number of dimensions grows, the data points may start to appear being more and more equally distant from one another. An intuitive explanation to the curse of dimensionality can be found for example in [21]. This phenomena can be tackled with various dimensionality reduction techniques, such as principal component analysis. A clear downside is that these techniques always result in a loss of some of the structure in the data.

3.3 Partitional methods

Given a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, partitional clustering methods attempt to group the data points into $k \leq n$ disjoint partitions, where each partition represents a cluster C_i for $i \in \{1, ..., k\}$. Typically partitional clustering methods attempt to minimize an objective of the form

$$\arg\min_{C_i} \sum_{k} \sum_{x \in C_i} d(\boldsymbol{x}, \hat{\boldsymbol{c}}_i) \tag{47}$$

where d is the distance measure and $\hat{\boldsymbol{c}}_i$ is the *cluster representative* of cluster C_i. This partition problem for distances such as the squared Euclidean distance is in fact NP-hard, but in the following sections we review two well-studied heuristics for this problem.

3.3.1 K-means algorithm

A traditional partitional clustering algorithm is the k-means algorithm [22]. Given an initial number of clusters k and a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, the k-means algorithm seeks to minimize the sum of squared residuals objective

$$\arg\min_{C_i} \sum_{k} \sum_{x \in C_i} \|\boldsymbol{x} - \hat{\boldsymbol{c}}_i\|_2^2 , \tag{48}$$

where $\hat{\boldsymbol{c}}_i$ is the *cluster centroid* of cluster C_i. Here the cluster centroid is equivalent to the mean vector of the cluster's members.

The k-means algorithm is guaranteed to converge to a local minimum. However, one problem with the local minimum is that there can be a number of local minima. In practice, a technique known as *random restarting* is commonly used to help finding a good minimum. In random restarting, the procedure is restarted once the algorithm has converged, and the best solution is returned among the executions of the algorithm. The time complexity of each iteration in the k-means algorithm is $\mathcal{O}(n \cdot k \cdot p)$.

The k-means++ algorithm [23] is a practical extension to the k-means algorithm that differs in its initialization strategy. The k-means++ algorithm uses a probabilistic initialization strategy, which attempts to maximize the initial distances between the cluster centroids. The procedure is simply as follows. One data points is picked uniformly at random as the cluster centroid of the first cluster. At the next step, the smallest distance $d(\boldsymbol{X}_i, \hat{\boldsymbol{c}}_j)$ to any of the cluster centroids $\hat{\boldsymbol{c}}_j$ is computed for all data points and subsequent cluster centroids are then chosen using weighted probabilities proportional to $d(\boldsymbol{X}_i, \hat{\boldsymbol{c}}_j)$.

There are a couple of practical issues with the k-means algorithm. For one, it is not at all obvious what the number of cluster k should be. This problem is not specific to the k-means algorithm but many clustering algorithms in general. Textbooks such as [20] review some approaches for choosing k. Secondly, k-means forces spherical clusters with similar variance between the clusters, even though this may not reflect the real structure of the data at all.

In Figure 6, we see the partitions of two synthetic data sets. On the left-hand side we see a successful partitioning of three spherical groups, but on the right-hand side the k-means algorithm is unable to capture the chain-like structure of the two data groups.

3.3.2 K-medoids algorithms

More robust partitional clustering methods include the implementations of the k-medoids approach. There are two main differences between the k-means algorithm and the implementations of the k-medoids approach. The first difference is the notion of cluster representatives. Whereas the k-means algorithm defines the cluster representative as the mean vector of the cluster's members (centroid), k-medoids

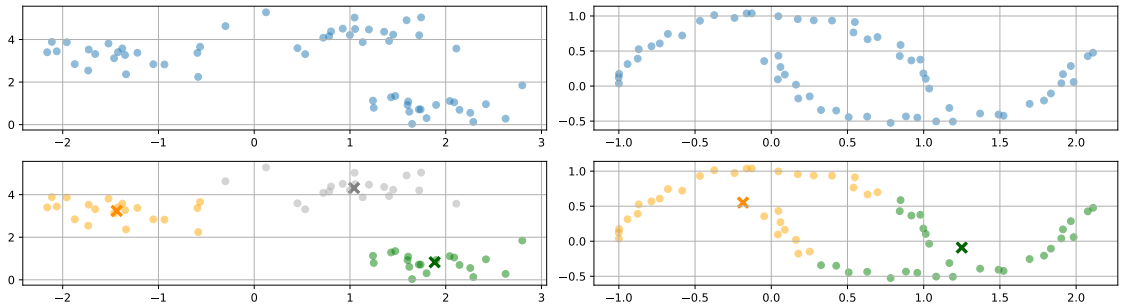

Figure 6: Top row: two artificially generated two-dimensional data sets. On the left-hand side, we see an example of a data set that would ideally form three clear clusters. These clusters are shown on the bottom left panel given by the k-means algorithm. On the right-hand side, we see two separable groups of points in shapes of a half circle. Here the k-means algorithm is unable to capture the clusters, as seen on the bottom right panel. The cluster centroids are shown as dark crosses.

uses a data point as the cluster representative, which is called a *medoid*. A centroid is easily affected by outliers in the cluster, whereas a medoid is not. However, this additional robustness does come with an increased computational cost. The second difference is that k-medoids is not restricted to the squared Euclidean distance measure but also works with arbitrary distance measures.

A classical implementation of a k-medoids clustering is the *partitioning around medoids* (PAM) algorithm [24]. PAM iteratively attempts to swap the current medoid into a better alternative. The time complexity of each iteration in the PAM algorithm is $\mathcal{O}(k(n-k)^2)$. Here the number of dimensions parameter p has been omitted as the distance matrix is assumed to have been computed in advance. In practice, PAM can be computationally intractable when both n and k are large.

A possible downside of both the k-means and k-medoids approaches comes from the objective being minimized. For example, minimizing the squared Euclidean distance to the cluster representative emphasizes compactness of the clusters, and not connectivity of cluster members.

3.4 Hierarchical methods

Hierarchical clustering methods, on the other hand, do not necessarily require the number of clusters to be initially parameterized. These clustering methods form a tree of clusters, where the tree known as a *dendrogram*. An advantage of the dendrograms is that they have nice visual representations, which may reveal meaningful taxonomies of the data points. Hierarchical clustering methods can be divided into *agglomerative* and *divisive* methods.

3.4.1 Agglomerative approach

Agglomerative clustering approach, also known as the bottom-up approach, is the more common hierarchical clustering strategy. In agglomerative clustering, each data point initially forms its own cluster. These clusters are then iteratively merged into one another based on some cluster closeness criteria. The iteration stops once every cluster has been merged into one cluster containing all the data points, or when some predefined threshold of the number of clusters is met. Agglomerative clustering is a greedy and deterministic[2] procedure.

Cluster closeness, which cluster to merge into another, is defined by *linkages*. Given sets of data points, we merge two sets A and B that has the smallest linkage value $d(A, B)$. Next we review some regularly used linkage functions.

Single linkage, also known as the nearest-neighbor technique, is defined as the minimum distance between the members of opposite clusters

$$d_{SL}(A, B) = \min_{a \in A,\, b \in B} d(a, b). \tag{49}$$

Single linkage computes the cluster similarity as the two most similar data points in opposite clusters while ignoring all other data points in the clusters — which means that the overall structure of the clusters is not taken into account. Single linkage is able to form cluster of complex shapes, but may be affected by aberrant data points between the clusters. For example, hierarchical agglomerative clustering with single linkage is able to capture the long chains of data points as seen in the bottom-left dendrogram in Figure 8.

On the opposite side, complete linkage is defined as the maximum distance between the members of opposite clusters

$$d_{CL}(A, B) = \max_{a \in A,\, b \in B} d(a, b). \tag{50}$$

The complete linkage can be thought to merge two clusters with the smallest diameter between the clusters.

Average linkage is defined as

$$d_{AL}(A, B) = \frac{1}{|A||B|} \sum_{a \in A,\, b \in B} d(a, b). \tag{51}$$

Similar to the average linkage, the mean linkage is defined as

$$d_{ML}(A, B) = d\left(\frac{1}{|A|} \sum_{a \in A} a,\ \frac{1}{|B|} \sum_{b \in B} b \right). \tag{52}$$

[2]In the strict sense, this may not always be true. Some agglomerative clustering implementations do not specify how to handle ties in the merging step, i.e. which clusters to merge when there is more than one pair of clusters that are equally distant from each other.

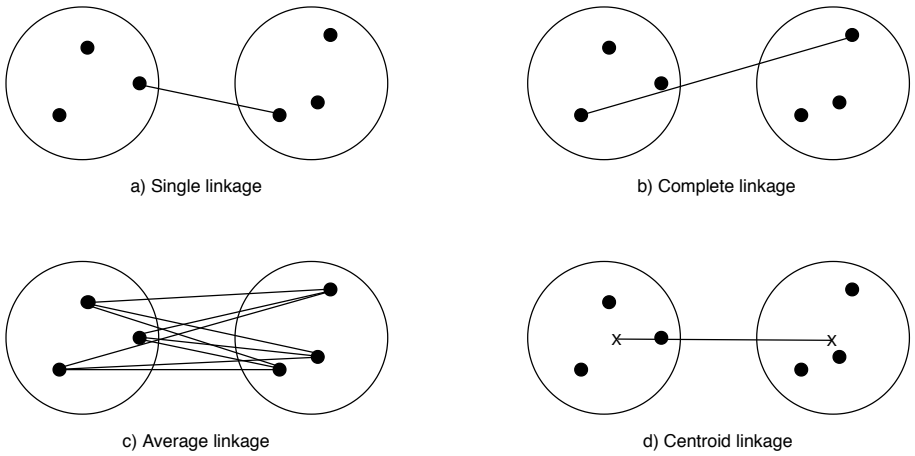

Figure 7: Different linkage functions used by hierarchical agglomerative clustering methods to merge two sets of data points.

Ward's method calculates the minimum increase in the sum of squared Euclidean distances to the cluster centroid for the merge $A \cup B$. Ward's method is defined as

$$d_W(A,B) = \sum_{x \in A \cup B} \|x - \mu_{A \cup B}\|_2^2 - \Big(\sum_{a \in A} \|a - \mu_A\|_2^2 + \sum_{b \in B} \|b - \mu_B\|_2^2 \Big) \tag{53}$$

$$= \frac{|A||B|}{|A| + |B|} \|\mu_A - \mu_B\|_2^2 , \tag{54}$$

where μ_i is the arithmetic mean of cluster i.

One particular downside of the hierarchical agglomerative clustering algorithms is their computative cost. Naive implementations, regardless of the linkage, have time complexity of $\mathcal{O}(n^3)$ and space complexity of $\mathcal{O}(n^2)$. In practice these implementations are typically never applicable to large data sets. With a priority queue this time complexity can be reduced down to $\mathcal{O}(n^2 \cdot \log n)$. For single linkage, there exists a method for reducing the time complexity down to $\mathcal{O}(n^2)$ and the space complexity down to $\mathcal{O}(n)$ [25]. Another downside of the hierarchical agglomerative clustering algorithms is the commitment to the split: once the merge is completed, the algorithm is unable rollback the merge even if it proves to be suboptimal in later iterations.

3.4.2 Divisive approach

Divisive (or top-down) clustering approach can be thought as the opposite of the agglomerative approach. Divisive clustering starts off by having every data point in a single cluster, which is recursively divided into smaller clusters. In practice this is often done by first applying a partitional clustering algorithm, such as the k-means algorithm, followed by recursively applying the partitional clustering algorithm on each cluster. An advantage with the divisive approach is that it is initially able to take the global distribution of the data points into account. On the other side, the

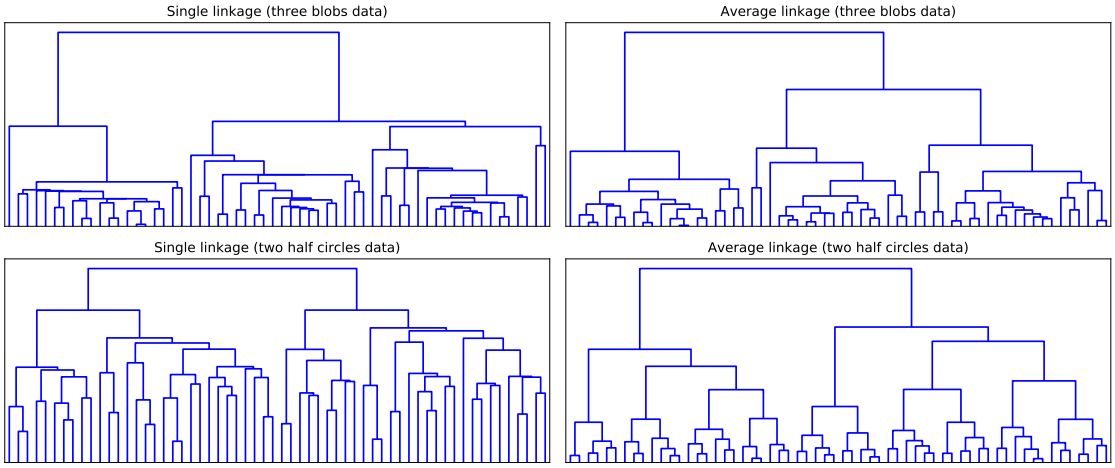

Figure 8: Dendrogram of the data shown in Figure 6. On top row we see the behavior of the single and complete linkages on the three blobs data. In the top right panel the average linkage shows three clear clusters. Respectively, on the bottom row we see the behavior of these linkages on the two circles data. In the bottom left panel the single linkage is able to capture two distinct clusters, which reflects the true structure of the two half circles. The vertical axes are on linear scale.

divisive approach suffers from all the same disadvantages as the partitional clustering methods.

3.5 Cluster evaluation

The most challenging part of a clustering process is arguably cluster evaluation. As we do not know the true labels of the data points, or that a reasonable labeling should even exist, it becomes highly subjective whether we have found reasonable structure in the data. There are two commonly used cluster evaluation measures: internal and external measures. In this section, we review some commonly used internal and external cluster evaluation measures.

3.5.1 Internal measures

Internal measures work directly on the clustering results and do not require any external labeling of the data. These measures take the clusters as input, and output a value, say between -1 and 1, where higher values denote better clusters in some sense. A clear disadvantage of the internal measures is that the function or objective measured could also be used as a clustering objective itself. Therefore an internal measure may not necessarily indicate whether the clusters are meaningful, but rather how well a particular clustering objective was solved [26].

Silhouette [27] computes a score of how similar a data point is to the points in its own cluster compared to the data points in other clusters. Let d be a distance metric

and i a data point of cluster C_k. We then define two functions, one for computing the similarity of data point i with the other data points in cluster C_k, and one for computing the similarity of i with the data points in other clusters. These functions, a and b, are defined as

$$a(i) = \frac{1}{|C_k| - 1} \sum_{j \in C_k, i \neq j} d(i, j) \tag{55}$$

$$b(i) = \min_{i \neq k} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j). \tag{56}$$

Now the *silhouette score* for data point i is computed as

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i),\, b(i)\}}, \tag{57}$$

where $|C_i| > 1$. For singleton clusters $|C_i| = 1$, we define $s(i) = 0$.

We see that data point i receives positive values when $b(i) > a(i)$, which can be interpreted as i being in its current cluster is the best assignment for i in the average sense given the clusters and distance metric d.

3.5.2 External measures

External measures validate the clustering results based on knowledge that was not used in the clustering task. This knowledge may include, if available, the ground truth, reasonable beliefs of the clusters or some external benchmarks. For example, we could validate the results using the true labels, attempt to label a subset of the data ourselves and then validate the results, or even consult a domain expert whether the clusters seem reasonable.

Cluster *purity* is a transparent measure that calculates the proportion of data points that were grouped correctly given the true labels of the data points. Purity is defined as

$$purity(C, T) = \frac{1}{N} \sum_{k} \max_{j} |c_k \cap t_j|, \tag{58}$$

where N is the number of the data points, k is the number of the clusters, $c_k \in C$ is a cluster of data points and $t_j \in T$ is a set of data points with the same label. Note that purity does not account for the number of clusters. For instance, setting every data point into its own cluster achieves a purity of 1.

Similarly, *rand index* [28] measures the percentage of correct cluster assignments to the ground truth. Rand index interprets the clusters as a series of pairwise decisions between the cluster assignments of two data points. It is defined as

$$RI = \frac{TP + TN}{TP + FP + TN + FN} = \frac{TP + TN}{\binom{N}{2}}, \tag{59}$$

where TP is the number of data point pairs that were correctly assigned into the same cluster with respect to the the ground truth (true positives), TN is the number of pairs that were correctly assigned into different clusters (true negatives), FP is the number of pairs that were incorrectly assigned into the same cluster (false positives) and FN is the number of pairs that were incorrectly assigned into different clusters (false negatives). In the latter equation, the denominator $\binom{N}{2}$ represents the total number of unordered pairs of N elements. Rand index gives equal weight to false positives and false negatives, which in some cases may be undesirable. Moreover, similar to purity, rand index also suffers from a large number of clusters as it only considers the correctly assigned pairs. An improvement to the rand index is the adjusted rand index [29].

Occasionally we consider putting similar objects into different clusters being worse than putting dissimilar objects into the same cluster. Whereas rand index gives equal weight to the false positives and false negatives, *f-measure* can be parameterized to penalize false negatives more than false positives by selecting $\beta > 1$. F-measure is defined as

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 + PR}, \tag{60}$$

where

$$P = \frac{TP}{TP + FP} \tag{61}$$

$$R = \frac{TP}{TP + FN}. \tag{62}$$

Values P and R are more generally known as precision and recall, respectively.

4 Time series clustering

Whereas the previous section introduced general clustering theory and approaches, this section delves into the more fine-grained area of time series clustering. In particular, time series clustering approaches are typically required to handle the temporal nature of the data. We focus especially on time series similarity, dimensionality reduction and clustering methodologies.

4.1 Similarity measures

As with any kind of measures of similarity, the choice of similarity for time series is heavily dependent on the requirements of the application. For example, if we were to estimate the price of a real estate, we could do this by estimating factors that have an influence on the price, such as the size, location and overall condition of the property. In this setting we can think of the factors being static and somewhat uncorrelated. However, observed time series can differ largely in length, frequency of the observations, contain distortions in amplitude and phase, and so on. Therefore it is essential to acknowledge various dynamics incorporated in time series.

For example, a problem with the Euclidean distance on temporal data can be that it treats the data points as if they were independent. In other words, if we were to permute the data points of two temporal data sequences, the Euclidean distance between the data points would remain unchanged. Sometimes this may be preferred, but often we want to account for the temporal correlation in the data.

Time series similarity can be split into roughly three categories: similarity in time, similarity in shape and similarity in change. Similarity in time measures similarity of the observations at corresponding points in time. An obvious example of similarity in time is the Euclidean distance. Another linear correlation based measure of similarity can be expressed as the Pearson correlation, which is defined as

$$d_{COR}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\sum_{t=1}^{T}(a_t - \bar{\boldsymbol{a}})(b_t - \bar{\boldsymbol{b}})}{\sqrt{\sum_{t=1}^{T}(a_t - \bar{\boldsymbol{a}})^2}\sqrt{\sum_{t=1}^{T}(b_t - \bar{\boldsymbol{b}})^2}}, \tag{63}$$

where $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$ denote the sample means of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. Pearson correlation computes a value in the interval $[-1, 1]$, where values close to 1 indicate a positive linear correlation, values close to 0 indicate no linear correlation and values close to -1 indicate a negative linear correlation.

Chouakria and Nagabhushan [30] proposed a similarity measure that can be seen as a slight modification of the Pearson correlation. This similarity measure addresses both the rate and direction of change as the first order temporal correlation coefficient. It is defined as

$$d_{CORT}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\sum_{t=1}^{T-1}(a_{t+1} - a_t)(b_{t+1} - b_t)}{\sqrt{\sum_{t=1}^{T-1}(a_{t+1} - a_t)^2}\sqrt{\sum_{t=1}^{T-1}(b_{t+1} - b_t)^2}}, \tag{64}$$

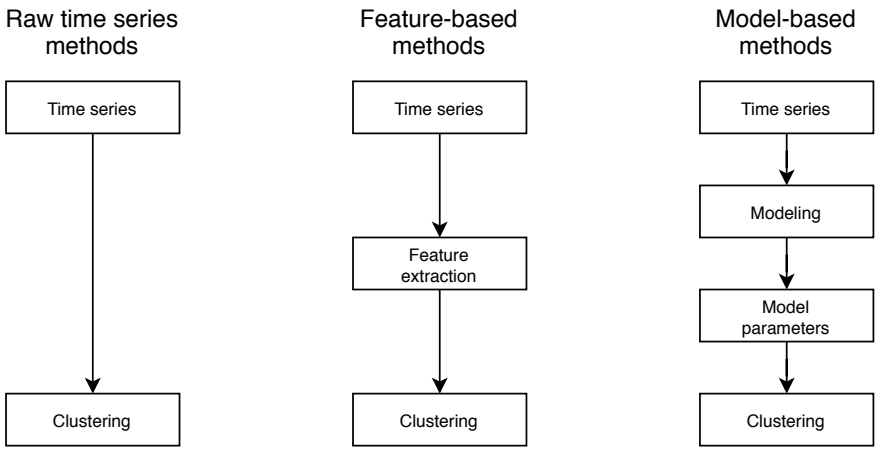

Figure 9: A high-level grouping of time series clustering methodologies.

where $d_{CORT}(\boldsymbol{a}, \boldsymbol{b})$ belongs to the interval $[-1, 1]$. Values close to 1 indicate similarity in dynamic behavior, such that the fluctuation in $\boldsymbol{a}$ and $\boldsymbol{b}$ at any point in time show similarity in direction and rate. Respectively, values close to -1 denote that the sequences behave conversely in direction and rate.

Another family of time series similarity is the similarity in shape. Similarity in shape gives more weight to the existence of similar patterns in the time series, rather than the similarity of the patterns in time. For example, dynamic time warping (DTW) [31] computes the global optimal alignment between two time series. DTW can be considered a generalization of methods for sequences of discrete values that compute the minimum number of operations to transform one sequence into another.

Time series databases are often massive in size. Therefore efficient methods to represent and process the time series must be emphasized. Bagnall et al. [32] proposed a clipping, or hard limiting, procedure for time series dimensionality reduction. Let μ be the mean of a time series. Their clipping procedure assigns a binary value of 1 for values greater than the mean μ, and a binary value 0 otherwise. This approach can be sufficient for time series with long flat periods including sudden peaks in the data. This technique also has a notable secondary advantage of compressing the original data into a very small space.

Piecewise aggregate approximation (PAA) [33] reduces the dimensionality of time series by splitting the input time series into chunks of equal length, followed by computing the mean of the values in every chunk. This mean then represents the values of the chunk. The symbolic aggregate approximaxtion (SAX) [34] takes the approach of PAA one step further. Assuming we have computed a PAA representation of a time series, the SAX representation is then given by denoting certain ranges of means by discrete symbols. Discrete fourier transforms (DFT) and related techniques have also received lots of attention in time series dimensionality reduction and pattern matching [35].

4.2 Clustering methodologies

Time series clustering approaches can be split into three main categories: raw time series methods, feature based methods and model based methods [36, 37]. A high-level layout behind these methodologies is seen in Figure 9. The obvious time series clustering approach is to work directly with the observed, raw time series. One novel shape-based time series clustering algorithm is the k-shape algorithm [38], which is designed to work directly on the observed time series. The k-shape algorithm is a partitional clustering algorithm that uses a normalized version of the cross-correlation measure to compute the cluster centroids.

Feature-based clustering methods extract characteristics from time series to be used as features for the clustering algorithms. This extraction of characteristics can also be considered as kind of a dimensionality reduction of the original data. Our application in Section 5 can be categorised as a feature-based clustering approach, where the extracted seasonality components are passed as input to the clustering algorithm. A thorough survey of feature-based clustering methods is presented in [18].

Model-based clustering methods attempt to fit a mathematical model, such as a hidden Markon Model or an ARIMA model to the observed data, and then cluster the data based on similarity of the fitted models' parameters [39]. This approach has been shown to be applicable only with long time series [18].

5 Application to the clustering of annual seasonality patterns

The motivation for this application stems from a crucial business requirement of improving sales forecasting accuracy in retail business. Sales time series are affected by various factors of which seasonality is the key factor in our study. We assume that identifying and forming groups of products with similar seasonal behavior would aid us in constructing more accurate sales forecasting models.

5.1 Overview of the application

In our application we attempt to form clusters of products that reflect the products' annual seasonality patterns. These seasonality patterns are identified from repeating characteristics in the sales data, such as peaks during the holiday seasons. The raw sales data is extremely noisy, so we must preprocess the data accordingly. Successful preprocessing is essential to accurately identify the seasonality from all other factors.

After we have preprocessed the data and identified the seasonality patterns, we form the clusters. These clusters are formed using common partitional and hierarchical clustering methods, which are represented in Section 5.4. Our cluster evaluation consists of two separate approaches. The first approach evaluates the clusters against prior labeling of a subset of the data. The second approach reflects the larger sales forecasting problem at hand: we create a simple forecast model based on the products' cluster's seasonality patterns. This forecast model is then compared against other validation models. Both of the cluster evaluation approaches are presented in Section 5.5. Results, evaluation and further discussion is presented in Section 6.

A closely related study was provided by Kumar et al. [5] who proposed a hierarchical clustering procedure with a distance function that accounts for measurement errors in the data. In their research they used retail sales data that had been preprocessed such that all non-seasonal sales factors were assumed to have been removed and the sales had been prior normalized to be on the same scale. Nevertheless, they found that their clustering procedure performed better than the k-means algorithm or a hierarchical agglomerative clustering algorithm with Ward's method when comparing the clusters against prior seasonality knowledge from merchants.

We too use external product information in the form of a *product tree*, but despite that, providing refined seasonality information given raw sales data is part of our pipeline. In general, a product tree is defined as a hierarchical structure, which reflects a natural taxonomy of the products. For example, one level of the product tree could include categories or nodes, such as meat, fish, bread and dairy. Similarly, the meat category could have subcategories or subnodes of different kind of meats, such as beef, pork and chicken. Physical structures of the stores themselves actually also reflect the hierarchy of the product trees: meats are typically organized into the same location in a store, and different kind of meats are mutually organized

within this location. Product categories are a pivotal reference of seasonality in our application — many products in the same category follow similar seasonality, but the ratio of products that do not, for instance, is a point of interest.

5.2 Weekly sales dataset

The sales data we analyze is a collection of sales time series from 50 stores of a single retail chain. The product set (3000 products) consists of products in a selection of product categories. This choice was simply guided by a decision to select a diverse subset of commonly sold items. These product categories are: alcoholic drinks, bread, chocolates, dairy, dried fruits, lamb, pork, root vegetables, salads and sweets.

Sales time series of individual products in individual stores are in the form of multivariate time series, which contains the monetary sales value and the sales price of the item sold on a weekly granularity. These time series can range from anywhere between the beginning of 2016 and the first quarter of 2020, or any combination of subranges during this period.

5.3 Preprocessing steps

As noted earlier it is not unusual that individual sales time series are very noisy. Some products are not sold throughout the year and some products sell with a very low frequency. Furthermore, sales are affected by various factors. These factors include but are not limited to price changes, campaigns, promotions, discounts, attributes of the product itself and even other products in the assortment. Preprocessing steps introduced in this section aim to diminish all the non-seasonal factors so that we can accurately estimate the seasonality factor.

5.3.1 Sales aggregation

Data aggregations are an essential concept when we want to summarize data. We perform two kind of sales aggregations in our application: one that reflects the products' sales globally and one that reflects the sales of groups of products with similar seasonality locally.

The first aggregation is a prerequisite for the smoothing procedure in the next section and it consists of summing the sales over all stores for every product, as shown in Figure 10. This aggregation strengthens the sales signal of individual products, especially for those products with sparse sales profiles in most stores. This aggregation also reduces the local bias of an individual store. Finally, to be able to capture the annually repeating patterns in the sales, all products whose aggregated sales do not contain a range of two consecutive years of sales are discarded from the analysis.

The second aggregation is a requirement for estimating local seasonal behavior in

individual stores for groups of products with similar global seasonality patterns for the forecast models in Section 5.5. Here we obtain aggregated sales time series for every product group and store combination, as seen in Figure 11.

5.3.2 Smoothing

We wish to filter out the high frequency peaks in the sales that can be identified with some certainty to be of non-seasonal cause, such as campaigns peaks. This filtering is done using a regularized weighted least squares smoothing procedure. Here the weights are formed under the assumption that most non-seasonal sales peaks show a negative correlation in sales prices — when we observe a notable reduction in the sales price, we often see a rapid peak in the sales volumes as well. Therefore we assign a low weight for the observations that show a reduction in sales price with respect to the sales prices in the recent past. One problem that this approach does not solve is that not all campaigns show a reduction in the sales price, but relate to other instruments to boost the sales, such as display stands at desirable locations inside the store.

Let $y_t = (v_t, p_t)$ denote the multivariate time series of the sales value v_t and sales price p_t at time t. The weights w_t for the weighted least squares objective are then calculated as follows:

$$w_t = \begin{cases} 0, & \text{if } 2 \cdot \frac{p_t}{P_t} - 1 < 0 \\ (2 \cdot \frac{p_t}{P_t} - 1)^2 & \\ 1, & \text{if } 2 \cdot \frac{p_t}{P_t} - 1 > 1, \end{cases} \tag{65}$$

where $P_t = \max(p_{t-6}, p_{t-5}, ..., p_t)$ is a causal filter of maximum sales prices in the recent past. For example, in Equation (65) a 50% reduction in sales price results in weight of 0, 25% reduction results in weight of 0.25 and 0% reduction results in weight of 1.

The smoothing objective that is minimized to estimate $\boldsymbol{x}$ is then defined as

$$\left\| \boldsymbol{W}^{1/2}(\boldsymbol{v} - \boldsymbol{x}) \right\|_2^2 + \lambda_1 \left\| \boldsymbol{D}_2 \boldsymbol{x} \right\|_2^2, \tag{66}$$

where $\boldsymbol{W} = \text{diag}(\boldsymbol{w})$, $\boldsymbol{v}$ is the sequence of observed sales values, $\lambda_1 = 4$ and $\boldsymbol{D}_2$ is the second-order difference matrix.

5.3.3 Decomposition

We use a robust parameterization of the STL decomposition procedure to accurately estimate the seasonality of a product. If we denote the length of the time series by N, our STL parameterization is then as follows: period length $n_{(p)} = 52$, number of inner loops $n_{(i)} = 1$, number of outer loops $n_{(o)} = 15$. Parameters for the three LOESS smoothers are: $n_{(s)} = 10N + 1$, $\lambda_{(s)} = 0$, $n_{(t)} = 79$, $\lambda_{(t)} = 1$, $n_{(l)} = 53$ and

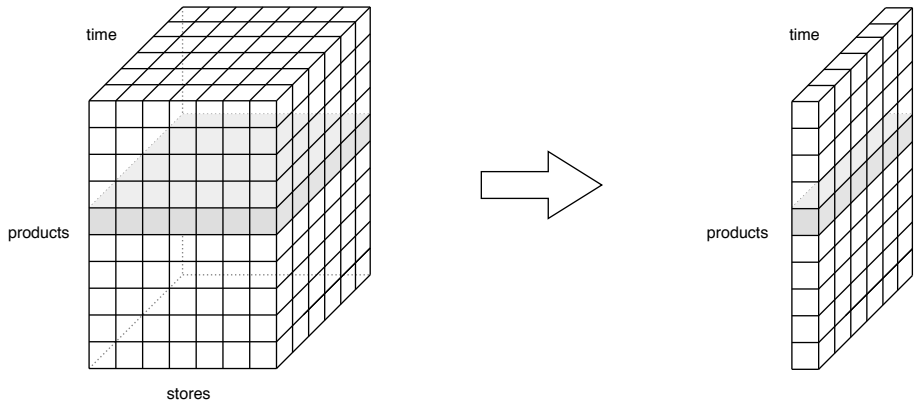

Figure 10: Product sales are summed over the stores as a preprocessing step to strengthen the sales signal and to reduce the local bias of the stores.

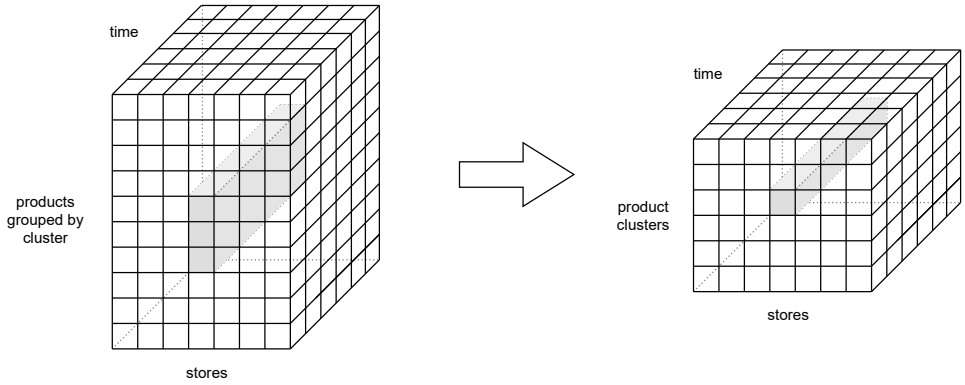

Figure 11: Here product sales are summed over groups of products and stores to obtain an aggregated sales time series for each product group and store combination.

$\lambda_{(l)} = 1$. Many of these parameters are the defaults suggested by the authors [7]. For a review of the parameters, we refer to Section 2.5.2.

After the decomposition, we pick the latest full year of the decomposed seasonality as the unnormalized feature vector $\boldsymbol{w} \in \mathbb{R}^{52}$ for the clustering algorithm. Here the components of $\boldsymbol{w}$ correspond to the weekly seasonality factors. Subsequently $\boldsymbol{w}$ is called the *seasonality prototype* or *seasonality weights*.

In our application a drawback of the STL method is its difficulty in estimating seasonality patterns of holidays and events that can occur at different times each year. For example, occurrence of the Easter holiday is deterministic, but from our perspective it can range from anywhere between March and April each year.

In Figure 12, we see a sample of four seasonality decompositions of the smoothed fits of sales aggregates. The STL decomposition is able to capture the annually repeating patterns with the exception of the decomposition shown on the bottom-left panel. This is an example product that has a sales peak during the Easter holidays, which explains the slight annual drift in the phase of the sales peaks.

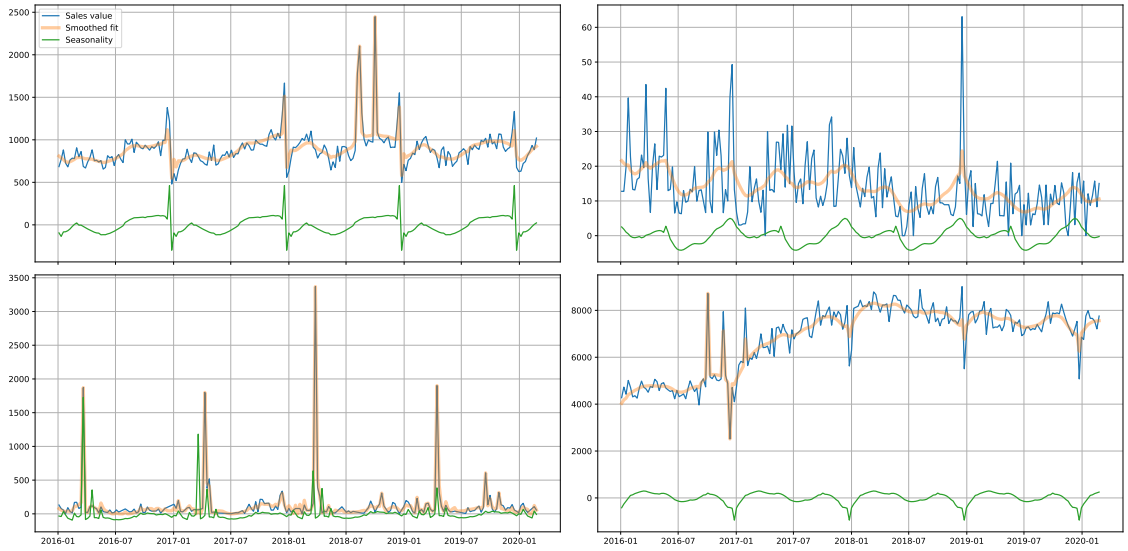

Figure 12: Four STL seasonality decompositions of the smoothed fits of sales values. On the vertical axis we have the weekly monetary sales values and on the horizontal axis we have the time of the observation.

5.3.4 Normalization

The decomposed seasonality can differ significantly in scale and amplitude. We chose to use z-normalization as it allows us to compare the seasonality prototypes with different amplitudes on the same scale. The z-normalization $\boldsymbol{z}$ of seasonality prototype $\boldsymbol{w}$ is defined as

$$\boldsymbol{z} = \frac{\boldsymbol{w} - \mu}{\sigma}, \tag{67}$$

where μ and σ are the mean and standard deviation of $\boldsymbol{w}$, respectively.

5.4 Clustering methods compared

Our cluster comparison consists of forming clusters of products using different clustering algorithms. These clusters are then evaluated with external benchmarks presented in the next section. The clustering algorithms compared in our analysis are the k-means algorithm and hierarchical agglomerative clustering algorithms with single, complete and Ward's linkages. The analysis is limited to predefined numbers of clusters: 10, 25, 50, 100 and 200. We use the Euclidean distance as our measure of dissimilarity in each clustering method.

Especially, we analyze which clustering algorithm and parameterization results in the minimum forecast error given the cluster seasonality forecast model. Structures of the clusters returned by these clustering algorithms are then analyzed in closer detail. For example, we attempt to find clusters that can be categorized by certain

events or time of the year — say a cluster that contains products that show a distinct sales peak at midsummer.

5.5 Cluster evaluation

We evaluate the clusters using two non-overlapping external evaluation approaches. In the first method we evaluate the clusters using prior beliefs of the products' sales. This is done by manually labeling subsets of products which we believe are likely to wind up into the same cluster given prior estimates of the product's seasonality. For example, we assume that most ice creams are more likely to peak in sales during the summer time, and are therefore likely to fall into the same cluster. The second evaluation method consists of a forecasting accuracy comparison between several forecasting models.

5.5.1 Manual labeling of products scores

In order to generate subjective ground truth, which we can compare against the clusters provided by the clustering algorithms, we must first manually label the data. We selected 5 subsets with 20 products in each subset that showed distinct seasonal characteristics. These characteristics include sales peaks or changes during different times of year and holidays. The following categories were identified from the data:

- *Easter*. Set of products that show a distinct sales peak particularly during the Easter holidays each year.
- *Summer*. Products that show a steady rise in sales towards summer followed by a slow decrease in sales towards winter.
- *Summer including midsummer peak*. Similar to the previous category with the additional distinct sales peak during midsummer.
- *Christmas*. These products only show a distinct sales peak during the Christmas holidays.
- *Early winter including Christmas peak*. Products that show a steady rise in sales from autumn to Christmas with the additional sales peak during the Christmas holidays.

5.5.2 External forecast benchmarks

In the second cluster evaluation, we compare a forecasting model that accounts for the cluster's seasonality prototype against four other forecasting models. Here the objective is to forecast a set amount of steps ahead of time given each model and compare their forecasting errors. The forecast models are trained on one year

(or equally 52 weeks) of observations and tested on the following H observations that were not used in training of the models. The granularity of the forecasts are the sales time series of individual products in individual stores. Here we selected 5000 sales time series at random as our forecast data. Let $\boldsymbol{y}$ denote one of these selections. Each $\boldsymbol{y}$ is split into a training sequence and a test sequence, where the training sequence corresponds to the components $y_1, ..., y_{52}$ and the test sequence corresponds to the components $y_{53}, ..., y_{52+H}$. Next we define the models used in our application to forecast the components $y_{53}, ..., y_{52+H}$.

1. *Cluster seasonality model.* The cluster seasonality forecast model assumes that products in the same cluster, in which the cluster's members should have similar seasonality patterns globally, also have similar seasonality patterns locally in individual stores. We aggregate the sales over every product cluster and store combination to obtain aggregated sales time series for these combinations, as presented in Section 5.3.1. The seasonality prototypes $\boldsymbol{w}$ for these combinations are then obtained by the smoothing and decomposition procedures presented in Sections 5.3.2 and 5.3.3, respectively.

   We then attempt to fit the product's cluster's seasonality prototype in the store we are forecasting in to the observed sales by minimizing the following sum of squared residuals objective:

   $$\arg\min_{a,b} \sum_{i=1}^{52} (y_i - (aw_i + b))^2. \tag{68}$$

   Here we should obtain a scalar a that scales the seasonality prototype $\boldsymbol{w}$ to match the fluctuation in $\boldsymbol{y}$ and a scalar b that shifts $\boldsymbol{w}$ to the level of $\boldsymbol{y}$. The forecast is then obtained by picking the first H components from $\hat{\boldsymbol{y}}$, which is defined as

   $$\hat{\boldsymbol{y}} = a\boldsymbol{w} + b. \tag{69}$$

2. *Mean model.* The mean forecast model computes the mean of $y_1, ..., y_{52}$ and simply repeats it H times ahead.

3. *Repeat previous year model.* The repeat previous year forecast model rewinds back a full year and takes a continuous segment of H values from corresponding points in that year. For $\boldsymbol{y} \in \mathbb{R}^{52+H}$, this model simply picks the first H components of $\boldsymbol{y}$ as the forecast.

4. *Product seasonality model.* The product seasonality model is similar to the cluster seasonality model except that it always uses the product's global seasonality prototypes and ignores the store we are forecasting in.

5. *Product tree seasonality model.* The product tree seasonality model is similar to the cluster seasonality model except that it uses the predetermined product categories as the clusters.

Note that none of these forecast models account for possibility of trend in the sales. This is actually preferred as we wish to only compare the forecasts from the perspective of matching seasonality.

In our application, the forecast errors are calculated as *mean absolute scaled errors* (MASE), which scales the *mean absolute errors* (MAE) of the forecast by the first-order differenced MAE of the training data. This scaling is preferable as it allows us to simultaneously compare forecast errors of different scales without additional parameterizations of the data. MASE is defined as

$$\text{MASE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{\frac{1}{H}\sum_{h=1}^{H} |y_{T+h} - \hat{y}_h|}{\frac{1}{T-1}\sum_{t=2}^{T} |y_t - y_{t-1}|}, \tag{70}$$

where $\boldsymbol{y}$ is the vector of observed sales values and $\hat{\boldsymbol{y}}$ is the forecast.

6 Results and discussion

In this section we provide the external validation scores for the selected clustering algorithms in our application. As presented in Section 5.4, four distinct clustering methods were compared. First we examine the f-measures, which are computed for the clusters given our manually labeled subsets of products. Then we review the forecast errors provided by the cluster seasonality models given the clusters of each clustering algorithm. Subsequently we evaluate the clusters provided by the clustering algorithms that yielded subjectively remarkable validation scores in closer detail.

6.1 Results

First we note the computational costs of the whole pipeline including the preprocessing and clustering steps. Overall the time of computation for each clustering algorithm, given the 3000 data points (products) with 52 dimensions (weeks), was negligible. Instead we spent most time in processing of the smoothing and decomposition procedures presented in Sections 5.3.2 and 5.3.3. These steps combined took one second on average for each time series per processor core. The preprocessing of 3000 time series then took $3000 \cdot 1s = 50$ minutes to compute. Especially when we preprocessed aggregated sales time series over the cluster and store combinations, the computational cost of these preprocessing steps started to accumulate. For example, given 50 stores and 200 clusters provided by a single clustering algorithm, the preprocessing took $50 \cdot \ 200 \cdot 1s \approx 2.75$ hours. In practice we were able to shrink the time of this computation to a fraction by parallelizing the workload into multiple processor cores.

Next we evaluate the f-measures ($\beta = 0.5$), which are shown in Table 1. These f-measures are obtained by matching the manually labeled subsets of data points to the clusters provided by the clustering algorithms. These manually labeled data points are from 5 distinct seasonality categories containing 20 labeled data points in each category. The choice of $\beta < 1$ reflects the intention to give more weight to precision than recall. As the number of clusters can be much larger than the number of manually labeled subsets, it is expected that the manually labeled subsets start to break down into more fine-grained clusters when the number of clusters increases. Therefore we want to penalize putting similar objects into different clusters less than we penalize putting dissimilar objects into the same cluster.

We see that the f-measure decreases for every clustering algorithm when the number of clusters parameter K increases, with the exception of hierarchical agglomerative clustering (HAC) with single linkage. Here the f-measures seem to be constant for every choice of K. A close inspection at the distribution of the cluster labels provided by HACs with single linkage actually reveals that given any number of clusters parameter K, the distribution of cluster labels is skewed towards one very large cluster and $K-1$ very small clusters, as illustrated in Figure 13. Therefore we

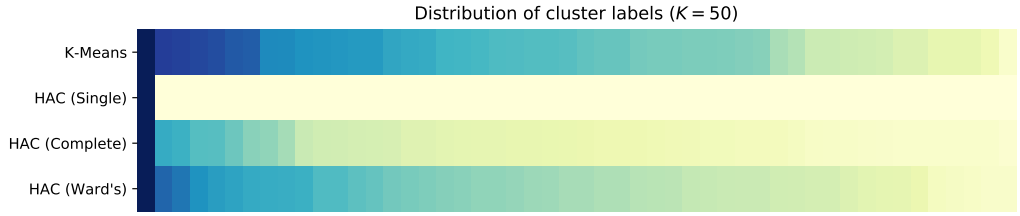

Figure 13: Distribution of cluster labels per clustering algorithms for the number of clusters $K = 50$. The left-most column with the darkest color row-wise indicate the cluster with maximum number of members given the set of clusters. The less dark cells within the row have been adjusted to be proportional in intensity with respect to this maximum.

F-measures ($\beta = 0.5$)

Algorithm	Number of clusters				
	10	25	50	100	200
K-Means	.738	.635	.547	.525	.385
HAC (Single)	.543	.543	.543	**.543**	**.543**
HAC (Complete)	.718	**.703**	.506	.495	.420
HAC (Ward's)	**.784**	.651	**.566**	.448	.345

Table 1: F-measures for $\beta = 0.5$.

cannot make reliable conclusions of the resulting clusters solely on the f-measures. For example, the k-means algorithm with initial number of clusters $K = 50$ has f-measure of 0.547, which is the closest value to the f-measures of HAC with single linkage among all choices of K. The clusters provided by the k-means algorithm may in fact turn out to be much more meaningful than the clusters provided by HAC with single linkage and any number of K.

Next we consider the external forecast benchmarks provided by the cluster seasonality models for each clustering algorithm and parameterization. The median errors given by these models are shown in Table 2. We see that the HAC algorithm with complete linkage and initial number of clusters $K = 200$ yielded the lowest median forecast error of 0.867. While ignoring HAC with single linkage, the highest median forecast error of 0.906 was provided by the HAC algorithm with complete linkage and $K = 10$. A common property of the k-means and HAC algorithms with complete and Ward's linkages was that the minimum forecast error was reached at the highest number of clusters $K = 200$. However, the differences in median forecast errors for this parameterization of K were not too far apart. Instead, the largest variance in forecast errors is seen with $K = 10$.

Standard deviations of the cluster seasonality model errors were 0.697 ± 0.010. At first they may seem unexpectedly large, but this observation actually reflects the nature of the data: we try to fit relatively smooth seasonality curves to raw sales

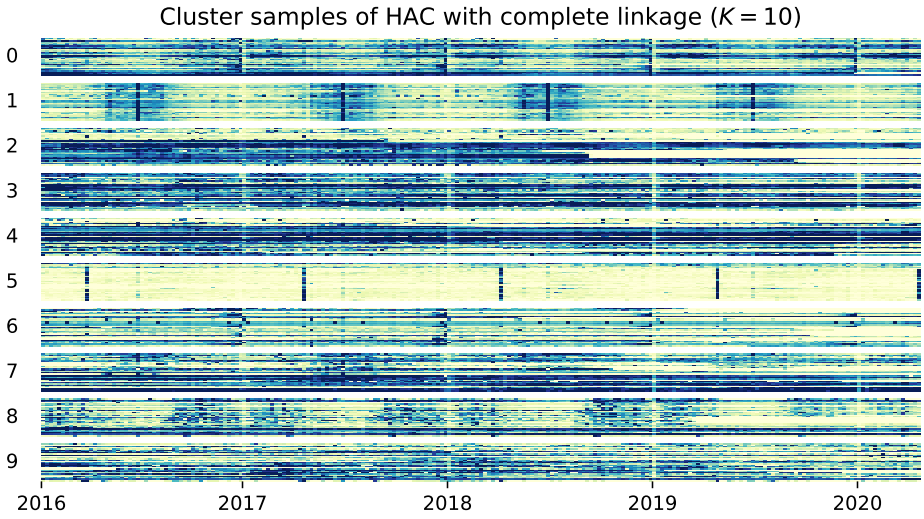

Figure 14: Thirty data samples from each cluster provided by the HAC algorithm with complete linkage and $K = 10$. Each row corresponds to a single sales time series. Sales values are represented as color codes, where darker colors reflect higher volumes in sales.

Median forecast errors

Algorithm	Number of clusters				
	10	25	50	100	200
K-Means	**.873**	**.880**	**.877**	.878	.872
HAC (Single)	.906	.907	.905	.901	.904
HAC (Complete)	.891	.882	.886	.876	**.867**
HAC (Ward's)	.884	.883	.880	**.875**	.870

Table 2: Median forecast MASEs provided by the cluster seasonality model for every clustering algorithm in our application. Linkages for the hierarchical agglomerative clustering algorithms (HAC) are presented in parenthesis. Every clustering method uses the Euclidean distance as its measure of dissimilarity. Standard deviations for the errors were 0.697 ± 0.010. Further analysis reveals that the behavior of HAC with single linkage is just not useful given our data.

time series that in most cases are wiggly and ragged. In other words, it is not unlikely that the forecast period contains some irregularities that the seasonality is simply unable to catch. Large deviations are also seen when we plot the 20th and 80th error percentiles of individual models later in this section.

For the rest of this section, we omit the results given by HAC with single linkage. Figure 14 shows thirty data samples from each cluster given by HAC with complete linkage and $K = 10$. In this figure the clusters are labeled by integers on the left-

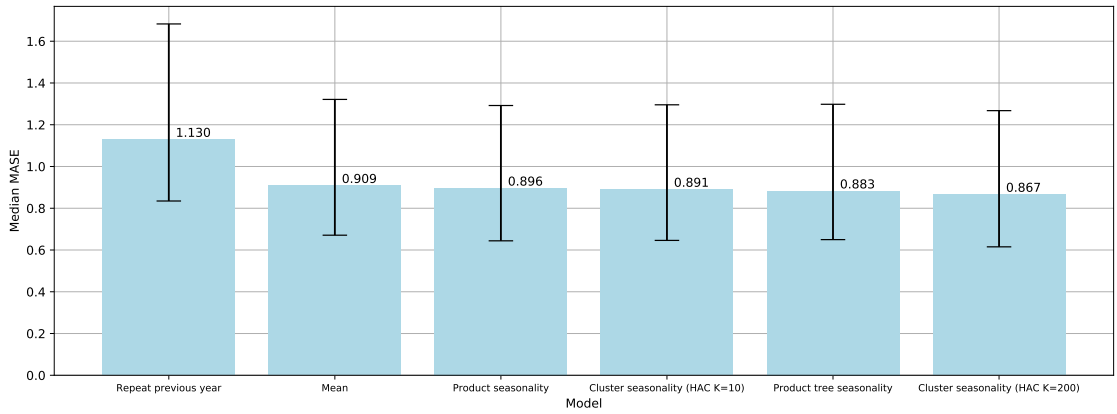

Figure 15: Median forecast MASEs of two cluster seasonality models and four validation models. The models are shown in descending order given the median forecast errors. Both HAC algorithms were parameterized with the complete linkage. The error bars represent the 20th and 80th percentiles.

hand side. The rows correspond to individual sales time series of a product and the sales volumes are presented in different colors. Here we can easily identify at least three distinct clusters with labels 0, 1 and 5. Cluster with label 0 contains products that sell mostly non-summer and cluster with label 1, on the other hand, contains products whose sales are concentrated on the summer time. Cluster with label 5 contains products that sell mostly during the Easter holidays.

When we compare the forecast benchmarks provided by the cluster seasonality models and product tree seasonality model, we see that many of the cluster seasonality model errors fall below the error of the product seasonality model, which was 0.883. Figure 15 arranges the validation models by their respective forecast errors in descending order. Importantly, we can observe several outcomes from this figure. All cluster seasonality models resulted in better forecast errors than the mean and product seasonality models. The cluster seasonality model resulting in better forecast errors than the mean model is only expected, but what is more notable is that all the cluster seasonality models resulted in better forecast errors than the product seasonality model, where the seasonality is obtained from the global sales aggregates of the product itself. This is in fact an indication that the minimum forecast error is not met when the number of clusters equals the number of data points, but instead that a reasonable clustering is more suitable than no clustering at all.

In Figure 16 we present the forecast errors of several seasonality models. Here the best cluster seasonality model in the sense of minimum forecast errors provided better forecasts than the product tree seasonality model in 9 product categories out of the 10 product categories. Especially, the best improvement in median forecast accuracy of +9% was achieved in the root vegetables product category. This result strengthens the intuition that many root vegetables have varying growing seasons; some are harvested late in the spring and some are harvested only towards autumn.

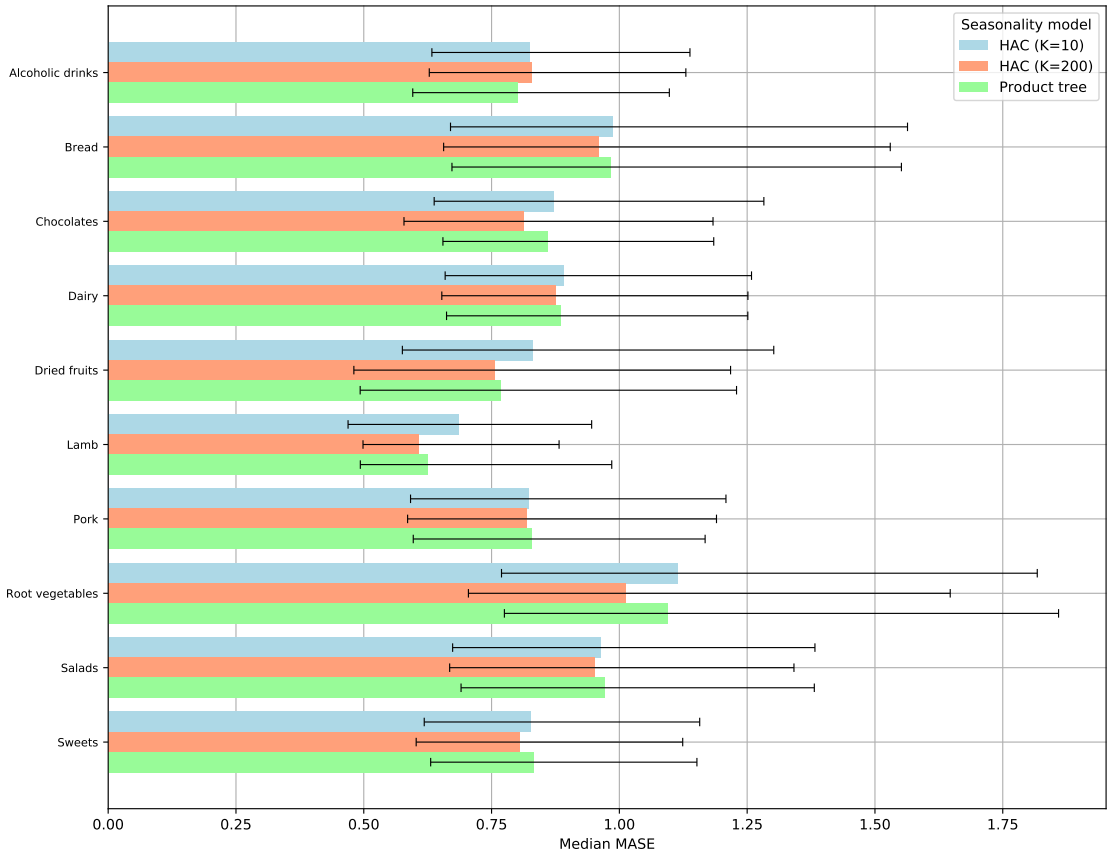

Figure 16: Median forecast MASEs of two cluster seasonality models and the product tree seasonality model per product category. Both HAC algorithms were parameterized with the complete linkage and initial numbers of clusters $K = 10$ and $K = 200$, respectively. The error bars represent the 20th and 80th percentiles.

6.2 Discussion

In the previous section we presented results suggesting that the clusters were able to capture some of the seasonal structure inside distinct product categories that the category itself does not represent. We saw how the k-means algorithm and hierarchical agglomerative clustering algorithms with complete linkage and Ward's method were able to form reasonable clusters with similar seasonality. These results provide a valid baseline for future research. For example, one point of interest would be to investigate what kind of result could be obtained with either dissimilarity measures that account for temporal correlation in the seasonality patterns or if we were to apply some kind of dimensionality reduction to the seasonality patterns prior to clustering. Next we conclude some important observations that were identified from the process as a whole.

A parameterization of the STL procedure with several iterations of the outer loop was found to be essential in successfully handling the irregularities in the smoothed

sales aggregates. Also applying the STL procedure on the aggregated sales without first smoothing the aggregates did not provide reliable seasonality patterns. The regularized weighted least squares smoothing procedure itself is an interesting point of research. It is likely that we could formulate an objective to process the aggregates even more subtly with additional information of the sales, such as incorporating campaign data in the weights.

The best clusters in terms of forecast accuracy were achieved with the number of clusters $K = 200$, which was the largest K in our parameterizations. Therefore it remains an open question whether the accuracy would improve if we were to increment K even further. Also practical limitations should be carefully considered with a large number of clusters.

In our application we also omitted the analysis of novelty products and any product that did not have two years of consecutive sales data. If available, utilizing product category information for these products to estimate seasonality does not seem like a bad option. Another option could be to estimate sales of products with sparse sales profiles based on their semantic features [6] followed by a seasonality decomposition.

One particular problem was identified in how the seasonality models estimate Easter sales. Forecasts of products specifically in the lamb category suffered from annual shifts of the holiday and should be interpreted with caution. For example, choosing the seasonality prototype of year 2019 to estimate Easter sales of 2016 is not the best option as the occurrence of Easter is more than three weeks apart during these years.

Furthermore, one possible flaw that may require attention lies in the annual decomposition of weekly data. Whereas it is straightforward to consider monthly seasonality as there is a fixed number of months in a year, the number of weeks in a year is not an integer 52, but instead a fraction 52.18. Estimating the number of weeks in a year to be exactly 52 causes a shift between data points that ought to be observed at corresponding times each year given that there are several years of data. However, in our domain this is not a major issue, as any item is more likely to contain too few sales than too many. In fact, seasonality prototype of the last season is likely to reflect the optimal seasonality of an item in the perspective of forecasting sales into the future, and we should not consider too many years into the past while estimating the present seasonality.

7 Conclusions

In this thesis we covered topics, such as general time series theory, time series smoothing and decomposition methods, and commonly used clustering approaches. In particular, our focus was on real world retail sales data, which is typically noisy and incomplete. We then provided a practical application, where we attempted to tackle these problems.

In this feature-based time series clustering application we formed clusters of products that reflected the products' annual seasonality patterns. We used classical smoothing and decomposition procedures on aggregated sales time series to estimate the seasonality patterns of retail products. These seasonality patterns were then used as input for the k-means and hierarchical agglomerative clustering algorithms. In particular, we were able to form clusters of products that better reflect the products' seasonality patterns than predefined groups of products, where the groups reflect the structure of how similar products are organized within close proximity in physical stores. We were indeed able to get smaller forecast errors with our seasonality forecast models when we used the clusters provided by the algorithms instead of using the predefined product groups.

Our research sets a viable starting point for future research on the area — whether the problem is simply to estimate seasonality of time series, validate if groups of time series follow similar seasonality or if the problem is to form groups of time series with similar seasonality. Whereas we focused on sales time series originating from the retail industry, approaches and methods used in this thesis need not to be retail-oriented, but should generalize equally well into other domains.

References

1 C. Peng, S. Havlin, H. E. Stanley, and A. L. Goldberger, "Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 5, pp. 82–87, 1995.

2 S. D. Campbell and F. X. Diebold, "Weather forecasting for weather derivatives," *Journal of the American Statistical Association*, vol. 100, no. 469, pp. 6–16, 2005.

3 T. Bollerslev *et al.*, "A conditionally heteroskedastic time series model for speculative prices and rates of return," *Review of economics and statistics*, vol. 69, pp. 542–547, 1987.

4 P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.

5 M. Kumar, N. Patel, and J. Woo, "Clustering seasonality patterns in the presence of errors," pp. 557–563, 2002.

6 A. Jha, S. Ray, B. Seaman, and I. Dhillon, "Clustering to forecast sparse time-series data," vol. 2015, 2015.

7 R. B. Cleveland and W. S. Cleveland, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, 1990.

8 A. Buja, T. Hastie, and R. Tibshirani, "Linear smoothers and additive models," *The Annals of Statistics*, pp. 453–510, 1989.

9 W. S. Cleveland and S. J. Devlin, "Locally weighted regression: an approach to regression analysis by local fitting," *Journal of the American statistical association*, vol. 83, pp. 596–610, 1988.

10 J. D. Hamilton, *Time series analysis*, vol. 2. Princeton New Jersey, 1994.

11 C. Chatfield and H. Xing, *The analysis of time series: an introduction with R*. CRC press, 2019.

12 R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2nd ed., 2018.

13 J. Shiskin, A. H Young, and J. C Musgrave, "The x-11 variant of the census method ii seasonal adjustment program," 1967.

14 V. Gomez and A. Maravall, "Programs tramo (time series regression with arima noise, missing observations, and outliers) and seats (signal extraction in arima time series). instructions for the user," *Documento de Trabajo*, vol. 9628, 1996.

15 R. Hafen, "Local regression models: Advancements, applications, and new methods," *ETD Collection for Purdue University*, 2011.

16 M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," pp. 449–460, 2005.

17 S. Parthasarathy, S. Mehta, and S. Srinivasan, "Robust periodicity detection algorithms," pp. 874—875, Association for Computing Machinery, 2006.

18 X. Wang, K. Smith-Miles, and R. Hyndman, "Characteristic-based clustering for time series data," *Data Mining and Knowledge Discovery*, vol. 13, pp. 335–364, 2006.

19 R. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, pp. 4–29, 1984.

20 J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

21 P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

22 J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, no. 14, pp. 281–297, 1967.

23 D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

24 L. Kaufman and P. J. Rousseeuw, "Partitioning around medoids (program pam)," *Finding groups in data: an introduction to cluster analysis*, vol. 344, pp. 68–125, 1990.

25 R. Sibson, "Slink: an optimally efficient algorithm for the single-link cluster method," *The computer journal*, vol. 16, no. 1, pp. 30–34, 1973.

26 R. Feldman, J. Sanger, *et al.*, *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

27 P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

28 W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

29 L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

30 A. Douzal and P. Nagabhushan, "Adaptive dissimilarity index for measuring time series proximity," *Advances in Data Analysis and Classification*, vol. 1, pp. 5–21, 2007.

31 H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

32 A. Bagnall and G. Janacek, "Clustering time series with clipped data," *Machine Learning*, vol. 58, pp. 151–178, 2005.

33 E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," 2001.

34 J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, 2003.

35 C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," *SIGMOD Rec.*, vol. 23, pp. 419—-429, 1994.

36 T. Liao, "Clustering time series data — a survey," *Pattern Recognition*, vol. 38, pp. 1857–1874, 2005.

37 S. Aghabozorgi, A. S. Shirkhorshidi, and T. Wah, "Time-series clustering - a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

38 J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," *SIGMOD Record*, vol. 45, pp. 69–76, 2016.

39 K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, pp. 273—-280, IEEE Computer Society, 2001.